Huang, Rubing and Chen, Jinfu and Towey, Dave and Chan, Alvin T.S. and Lu, Yansheng (2015) Aggregate-strength interaction test suite prioritization. Journal of Systems and Software, 99 . pp. 36-51. ISSN 0164-1212

Aggregate-Strength Interaction Test Suite Prioritization

Rubing Huang[a,b,*], Jinfu Chen[a], Dave Towey[c], Alvin T. S. Chan[d], Yansheng Lu[b]

[a]*School of Computer Science and Telecommunication Engineering, Jiangsu University, Zhenjiang, Jiangsu 212013, P.R. China*
[b]*School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, Hubei 430074, P.R. China*
[c]*School of Computer Science, The University of Nottingham Ningbo China, Ningbo, Zhejiang 315100, P.R. China.*
[d]*Department of Computing, The Hong Kong Polytechnic University, Hong Kong, P.R. China*

Abstract

Combinatorial interaction testing is a widely used approach. In testing, it is often assumed that all combinatorial test cases have equal fault detection capability, however it has been shown that the execution order of an interaction test suite's test cases may be critical, especially when the testing resources are limited. To improve testing cost-effectiveness, test cases in the interaction test suite can be prioritized, and one of the best-known categories of prioritization approaches is based on "*fixed-strength prioritization*", which prioritizes an interaction test suite by choosing new test cases which have the highest uncovered interaction coverage at a fixed strength (level of interaction among parameters). A drawback of these approaches, however, is that, when selecting each test case, they only consider a fixed strength, not multiple strengths. To overcome this, we propose a new "*aggregate-strength prioritization*", to combine interaction coverage at different strengths. Experimental results show that in most cases our method performs better than the test-case-generation, reverse test-case-generation, and random prioritization techniques. The method also usually outperforms "*fixed-strength prioritization*", while maintaining a similar time cost.

Keywords:
Software testing, combinatorial interaction testing, test case prioritization, interaction coverage, fixed-strength prioritization, aggregate-strength prioritization, algorithm

1. Introduction

Combinatorial interaction testing [29], is a black-box testing method that has been well researched, and applied in the testing of practical systems [14, 24, 42]. It focuses on constructing an effective test suite (called an interaction test suite) in order to catch failures triggered by the interactions among k parameters of the software under test (SUT). Here, parameters may represent any factors that affect the running of the SUT, such as user inputs, configuration options, etc., and each parameter may have several valid values. In fact, combinatorial interaction testing provides a trade-off between testing effectiveness and efficiency, because it only requires coverage of certain key combinations, rather than of all possible combinations, of parametric values. For instance, τ-wise ($1 \leq \tau \leq k$) combinatorial interaction testing, where τ is referred to as the level of interaction among parameters (named strength), constructs an interaction test suite to cover all possible τ-tuples of parameter values (referred to as τ-wise parameter value combinations).

Due to limited testing resources in practical applications where combinatorial interaction testing is used, for example in combinatorial interaction regression testing [32], the execution order of combinatorial test cases can be critical, and therefore the potentially failure-revealing test cases in an interaction test suite should be executed as early as possible. In other words, a well-ordered test case execution may be able to detect failures earlier, and thus enable earlier fault characterization, diagnosis and correction [29]. To improve testing efficiency, interaction test suites can be prioritized [29].

The prioritization of interaction test suites has been well studied [1, 2, 4–7, 18, 30–33, 37, 39, 40], with many techniques having been proposed, such as random prioritization [1] and branch-coverage-based prioritization [32]. A well-studied category of prioritization approaches for interaction test suites is "*fixed-strength prioritization*", which prioritizes the interaction test suite by repeatedly choosing an unexecuted test case from candidates such that it covers the largest number of uncovered parameter value combinations at a fixed strength [1, 2, 4–7, 18, 30–33, 37, 39, 40]. However, when selecting each unexecuted test case, this strategy only considers interaction coverage of a fixed strength τ, rather than interaction coverage of multiple strengths: Although it focuses on τ-wise interaction coverage, it may neglect λ-wise ($1 \leq \lambda < \tau$)[1] interaction coverage when choosing the next test

*Corresponding author at: School of Computer Science and Telecommunication Engineering, Jiangsu University, 301 Xuefu Road, Zhenjiang, Jiangsu 212013, P.R. China.

Email addresses: rbhuang@mail.ujs.edu.cn, or rbhuang@hust.edu.cn (Rubing Huang), jinfuchen@ujs.edu.cn (Jinfu Chen), dave.towey@nottingham.edu.cn (Dave Towey), cstschan@comp.polyu.edu.hk (Alvin T. S. Chan), lys@hust.edu.cn (Yansheng Lu)

[1] For ease of description, in this paper we assume τ is a constant, because τ is obtained from an interaction test suite; while λ is a variable where $1 \leq \lambda \leq \tau$.

case. Consequently, "*fixed-strength prioritization*" may not use sufficient information to guide the prioritization of the interaction test suite — an example of this will be given in the following section.

To evaluate the difference between a combinatorial test case and the already executed test cases, we propose a new dissimilarity measure which considers different interaction coverage at different strengths. Based on this, we present a heuristic algorithm which, given an interaction test suite T of strength τ, chooses an element from among candidates after comprehensively considering different interaction coverage at strengths from 1 to τ, and assigning each interaction coverage a weight. The method gives a priority of all strengths from 1 to τ, and balances λ-wise interaction coverage for $\lambda = 1, 2, ..., \tau$. This proposed method has the advantage over existing prioritization methods by employing more information to guide the prioritization process. We refer to this method as "*aggregate-strength prioritization*".

In terms of the rates of covering parameter value combinations and fault detection, experimental results show that in most cases our method performs better than the test-case-generation, reverse test-case-generation, and random prioritizations; and also has better performance than the "*fixed-strength prioritization*," while maintaining a similar time cost.

This paper is organized as follows: Section 2 introduces some background information, including combinatorial interaction testing, and test case prioritization. Section 3 describes some related work. Section 4 introduces a motivating example, and then proposes a new prioritization strategy, with an analysis of its properties and time complexity. Section 5 presents some simulations and experiments with real-life programs related to the use of the proposed strategy, and finally, Section 6 presents the conclusions and discusses potential future work.

2. Background

In this section, some fundamental aspects of combinatorial interaction testing and test case prioritization are presented.

2.1. Combinatorial interaction testing

Combinatorial interaction testing is used to generate a test suite to detect faults triggered by interactions among parameters in the SUT. For convenience, in the remainder of this paper we will refer to a *combination of parameters* as a *parameter combination*, and a *combination of parametric values* or a *parameter value combination* as a *value combination*.

Definition 1. *A test profile, denoted as* $TP(k, |V_1||V_2|\cdots|V_k|, \mathscr{D})$, *has information about a combinatorial test space of the SUT, including k parameters,* $|V_i|$ $(i = 1, 2, \cdots, k)$ *values for the i-th parameter, and constraints* $\mathscr{D}$ *on value combinations.*

Table 1: A Test Profile for a SUT

Parameter	p_1	p_2	p_3	p_4
Value	0	3	6	8
	1	4	7	9
	2	5	-	-

Table 1 gives an example of a SUT with $\mathscr{D} = \emptyset$, in which there are four parameters, two of which have two values and another two of which have three values: the test profile can be written as $TP(4, 2^2 3^2, \emptyset)$.

Definition 2. *Given a test profile* $TP(k, |V_1||V_2|\cdots|V_k|, \mathscr{D})$, *a k-tuple* $(v_1, v_2, \cdots, v_k)$ *is a combinatorial test case for the SUT, where* $v_i \in V_i$ $(i = 1, 2, \cdots, k)$.

For example, $(0, 3, 6, 8)$ is a 4-tuple combinatorial test case for the SUT shown in Table 1.

Definition 3. *The number of parameters required to trigger a failure is referred to as the failure-triggering fault interaction (FTFI) number.*

The combinatorial input domain fault model assumes that failures are caused by parameter interactions. For example, if the SUT shown in Table 1 fails when both p_2 is set to 5 and p_3 is set to 6, this failure is caused by the parameter interaction (p_2, p_3), and therefore, the FTFI number is 2.

In combinatorial interaction testing, a covering array is generally used to represent an interaction test suite.

Definition 4. *Given a* $TP(k, |V_1||V_2|\cdots|V_k|, \mathscr{D})$, *an* $N \times k$ *matrix is a* τ*-wise* $(1 \le \tau \le k)$ *covering array, denoted* $CA(N; \tau, k, |V_1||V_2|\cdots|V_k|)$, *which satisfies the following properties: (1) each column* i $(i = 1, 2, \cdots, k)$ *contains only elements from the set* V_i*; and (2) the rows of each* $N \times \tau$ *sub-matrix cover all* τ*-wise value combinations from the* τ *columns at least once.*

Table 2 shows an example covering array for the SUT in Table 1. The covering array, denoted as $CA(9; 2, 4, 2^2 3^2)$, only requires a set of nine test cases in order to cover all 2-wise value combinations.

Each column of a covering array represents a parameter of the SUT, while each row represents a combinatorial test case. Testing with a τ-wise covering array is called τ-wise

Table 2: $CA(9; 2, 4, 2^2 3^2)$ for the $TP(4, 2^2 3^2, \emptyset)$ shown in Table 1

Test No.	p_1	p_2	p_3	p_4
tc_1	0	3	7	9
tc_2	0	4	6	8
tc_3	0	5	7	8
tc_4	1	3	6	9
tc_5	1	4	7	8
tc_6	1	5	6	9
tc_7	2	3	7	8
tc_8	2	4	6	9
tc_9	2	5	6	8

combinatorial interaction testing. In this paper, we focus on τ-wise covering arrays, rather than on other interaction test suites such as *variable-strength covering array* [9].

In τ-wise combinatorial interaction testing, the *uncovered λ-wise value combinations distance* ($UVCD_\lambda$) is a distance (or dissimilarity) measure often used to evaluate combinatorial test cases against an interaction test suite [20].

Definition 5. *Given an interaction test suite T, strength λ, and a combinatorial test case tc, the uncovered λ-wise value combinations distance ($UVCD_\lambda$) of tc against T is defined as:*

$$UVCD_\lambda(tc, T) = |CombSet_\lambda(tc) \setminus CombSet_\lambda(T)|, \quad (1)$$

where $CombSet_\lambda(tc)$ is the set of all λ-wise value combinations covered by tc, and $CombSet_\lambda(T)$ is the set covered by all of T. More specifically, these can be respectively written as follows:

$$CombSet_\lambda(tc) = \{(v_{j_1}, v_{j_2}, \cdots, v_{j_\lambda}) | 1 \le j_1 < j_2 < \cdots < j_\lambda \le k\}, \quad (2)$$

$$CombSet_\lambda(T) = \bigcup_{tc \in T} CombSet_\lambda(tc). \quad (3)$$

In the past, minimization of the interaction test suite size has been emphasized in order to achieve the desired coverage, and although the problem of constructing interaction test suites (covering array and variable-strength covering array) is NP-Complete [35], many strategies for building them have been developed, including approaches employing greedy, recursive, heuristic search, and algebraic algorithms and methods (see [29] for more details).

2.2. Test case prioritization

Suppose $T = \{tc_1, tc_2, \cdots, tc_N\}$ is a test suite containing N test cases, and $S = \langle s_1, s_2, \cdots, s_N \rangle$ is an ordered set of T, called a test sequence, where $s_i \in T$ and $s_i \neq s_j$ $(i, j = 1, 2, \cdots, N; i \neq j)$. If $S_1 = \langle s_1, s_2, \cdots, s_m \rangle$ and $S_2 = \langle q_1, q_2, \cdots, q_n \rangle$ are two test sequences, we define $S_1 \succ S_2$ as $\langle s_1, s_2, \cdots, s_m, q_1, q_2, \cdots, q_n \rangle$; and $T \setminus S$ as the maximal subset of T whose elements are not in S.

Test case prioritization is used to schedule test cases in an order, so that, according to some criteria (e.g. condition coverage), test cases with higher priority are executed earlier in the testing process. A well-prioritized test sequence may improve the likelihood of detecting faults earlier, which may be especially important when testing with limited test resources. The problem of test case prioritization is defined as follows [34]:

Definition 6. *Given (T, Ω, g), where T is a test suite, Ω is the set of all possible test sequences obtained by ordering test cases of T, and g is a function from a given test sequence to an award value, the problem of test case prioritization is to find an $S \in \Omega$ such that:*

$$(\forall S')\,(S' \in \Omega)\,(S' \neq S)\,[g(S) \ge g(S')]. \quad (4)$$

In Eq. (4), g is a function to evaluate a test sequence S by returning a real number. A well-known function is *a weighted average of the percentage of faults detected* (APFD) [13], which is a measure of how quickly a test sequence can detect faults during the execution (that is, the rate of fault detection). Let T be a test suite containing N test cases, and let F be a set of m faults revealed by T. Let SF_i be the number of test cases in test sequence S of T that are executed until detecting fault i. The APFD for test sequence S is given by the following equation from [13]:

$$APFD = 1 - \frac{SF_1 + SF_2 + \cdots + SF_m}{N \times m} + \frac{1}{2N}. \quad (5)$$

The APFD metric, which has been used in practical testing, has a range of (0, 1), with higher values implying faster rates of fault detection. Two requirements of the APFD metric are that (1) all test cases in a test sequence should be executed; and (2) all faults should be detected. In practical testing applications, however, it may be that only part of the test sequence is run, and only some of the faults detected. In such cases, the APFD may not be an appropriate evaluation of the fault detection rate. To alleviate the difficulties associated with these two requirements, Qu *et al.* [32] have presented a new metric, *Normalized APFD* (NAPFD), as an enhancement of APFD, and defined it as follows:

$$NAPFD = p - \frac{SF_1 + SF_2 + \cdots + SF_m}{N' \times m} + \frac{p}{2N'}, \quad (6)$$

where m and SF_i $(i = 1, 2, \cdots, m)$ have the same meaning as in APFD; p represents the ratio of the number of faults identified by selected test cases relative to the number of faults detected by the full test suite; and N' is the number of the executed test cases. If a fault f_i is never found, then $SF_i = 0$. If all faults can be detected, and all test cases are executed, NAPFD and APFD are identical, with $p = 1.0$ and $N' = N$.

Many test case prioritization strategies have been proposed, such as fault severity based prioritization [12], source code based prioritization [34, 41], search based prioritization [26], integer linear programming based prioritization [44], XML-manipulating prioritization [28], risk exposure based prioritization [43], system-level test case prioritization [36], and history based prioritization [21]. Most prioritization strategies can be classified as either meta-heuristic search methods or greedy methods [40].

3. Related Work

According to Qu's classification, the prioritization of interaction test suites can generally be divided into two categories: (1) *pure prioritization*: re-ordering test cases in the interaction test suite; and (2) *re-generation prioritization*: considering the prioritization principle during the process of interaction test suite generation, that is, generating (or constructing) an interaction test sequence [32]. The method proposed in this paper, as well as the methods used for comparison, belongs to the former category. However, if based on the same prioritization principle, *pure prioritization* works in a similar manner to *re-generation prioritization*. For

example, when testers base prioritization on test case execution time, *pure prioritization* chooses an element from the given test suite such that it has the lowest execution time, and *re-generation prioritization* selects (or generates) such elements from the exhaustive test suite (or constructed candidates). In this section, therefore, we do not distinguish between *pure* and *re-generation* prioritization.

Bryce and Colbourn [1, 2] initially used four test case weighting distributions to construct interaction test sequences with seeding and constraints, which belongs to the *re-generation prioritization* category. Bryce *et al.* [4, 5] proposed a *pure prioritization* method without considering any other factors (only considering 2-wise and 3-wise interaction coverage) for interaction test suites, and then applied it to the event-driven software. Similarly, Qu *et al.* [31–33] applied test case weight to the *pure prioritization* method, and applied this method to configurable systems. They also proposed other test case weighting distributions based on code coverage [32], specification [32], fault detection [31], and configuration change [31]. Additionally, Chen *et al.* [7] used an ant colony algorithm to generate prioritized interaction test suites which considered interaction coverage information.

Srikanth *et al.* [37] took the cost of installing and building new configurations into consideration for helping prioritize interaction test suites. Bryce *et al.* [6] used the length of the test case to represent its cost, and combined it with pair-wise interaction coverage to guide the prioritization of interaction test suites. Wang *et al.* [40] combined test case cost with test case weight to prioritize interaction test suites, and also extended this method from lower to higher strengths, and proposed a series of metrics which have been widely used in the evaluation of interaction test sequences. Petke *et al.* [30] researched the efficiency and fault detection of the *pure prioritization* method proposed in [4, 5] with other (lower and higher) strengths. Recently, Huang *et al.* [18] investigated adaptive random test case prioritization for interaction test suites using interaction coverage, a method which, by replacing the prioritization method in [4, 5] attempts to reduce time costs, while maintaining effectiveness.

Throughout the interaction test suite prioritization process, the strategies so far mentioned [1, 2, 4–7, 18, 30–33, 37, 40] do not vary the strength of interaction coverage. For example, given a strength τ for prioritization, these prioritization strategies only consider τ-wise interaction coverage to guide the test cases selection. These implementations of "*fixed-strength prioritization*" are also referred to as *interaction coverage based prioritization* (or ICBP). Previous studies also investigated incremental strengths to prioritize interaction test suites. For instance, Wang [39] used incremental strengths, and proposed a *pure prioritization* method named inCTPri used to prioritize covering arrays. More specifically, given a τ-wise covering array $CA(N; \tau, k, |V_1||V_2|\cdots|V_k|)$, the inCTPri firstly uses interaction coverage at a low strength (such as λ where $1 < \lambda \leq \tau$) to prioritize CA; when all λ-wise value combinations have been covered by selected test cases, the inCTPri increases λ to $\lambda + 1$, and then repeats the above process until $\lambda > \tau$. In other words, inCTPri is actually ICBP using different strengths during the prioritization process. Huang *et al.* [19] proposed two pure prioritization methods for variable-strength covering arrays which exploit the main-strength and sub-strengths of variable-strength covering arrays to guide prioritization.

To date, most interaction test suite prioritization strategies belong to the category of "*fixed-strength prioritization*", because they only consider a fixed strength when selecting each combinatorial test case from candidates. Few studies have been conducted on the prioritization of interaction test suites using "*aggregate-strength prioritization*", and our study is, to our best knowledge, the first to use multiple strengths when choosing each combinatorial test case from the candidate elements.

4. Aggregate-Strength Interaction Test Suite Prioritization

In this section, we present a motivating example to illustrate the shortcoming of "*fixed-strength prioritization*", and then introduce a new dissimilarity measure for evaluating combinatorial test cases, the *weighted aggregate-strength test case dissimilarity* (*WASD*). We then introduce a heuristic algorithm for prioritizing an interaction test suite based on the *WASD* measure ("*aggregate-strength prioritization*" strategy), investigate some of its properties, and give a time complexity analysis.

4.1. A motivating example

Given covering array $CA(9; 2, 4, 2^2 3^2)$, shown in Table 2, "*fixed-strength prioritization*" generally uses strength $\tau = 2$ to guide the prioritization. More specifically, this strategy chooses the next test case such that it covers the largest number of 2-wise value combinations that have not yet been covered by the already selected test cases (that is, $UVCD_2$). We assume that "*fixed-strength prioritization*" is deterministic, e.g. the first candidate is selected as the next test case in situations with more than one best element, and therefore its generated interaction test sequence would be $S_1 = \langle tc_1, tc_2, tc_6, tc_5, tc_7, tc_8, tc_3, tc_4, tc_9\rangle$. Intuitively speaking, S_1 would face a challenge when a fault f_1 is triggered by "P_1=2", because it needs to run five test cases ($5/9 = 55.56\%$) in order to detect this fault. However, if multiple strengths were used to prioritize this interaction test suite, e.g. strengths 1 and 2, both 1-wise and 2-wise value combinations would be considered, and therefore we would obtain the interaction test sequence $S_2 = \langle tc_1, tc_9, tc_4, tc_2, tc_5, tc_7, tc_8, tc_3, tc_6\rangle$. S_2 would only require 2 test cases ($2/9 = 22.22\%$) to be run to identify the fault f_1, which means that S_2 has a faster fault detection than S_1.

Motivated by this, it is reasonable to consider different strengths for prioritizing interaction test suites, which may provide better effectiveness (such as fault detection) than "*fixed-strength prioritization*".

4.2. Weighted aggregate-strength test case dissimilarity

Given an interaction test suite T on $TP(k, |V_1||V_2|\cdots|V_k|, \mathscr{D})$, a combinatorial test case tc, and the strength τ, the *weighted aggregate-strength test case dissimilarity* (*WASD*) of tc against T is defined as follows:

$$WASD(tc, T) = \sum_{\lambda=1}^{\tau}(\omega_\lambda \times \frac{|UVCD_\lambda(tc, T)|}{C_k^\lambda}), \quad (7)$$

where $0 \le \omega_\lambda \le 1.0$ ($\lambda = 1, 2, \cdots, \tau$), and $\sum_{\lambda=1}^{\tau} \omega_\lambda = 1.0$.

Intuitively speaking, $WASD(tc, T) = 0$, if and only if $tc \in T$; $WASD(tc, T) = 1.0$, if and only if any 1-wise value combination covered by tc is not covered by T. Therefore, the *WASD* ranges from 0 to 1.0.

Here, we present an example to briefly illustrate the *WASD*. Considering the combinatorial test cases in Table 2, suppose interaction test sequence $S = \{tc_1\}$, strength $\tau = 2$, two candidates tc_2 and tc_9, and $\omega_1 = \omega_2 = \cdots = \omega_\tau = 1/\tau$, the *WASD* of tc_2 against T is $WASD(tc_2, S) = \frac{1}{2} \times \frac{UVCD_{\lambda=1}(tc_2,S)}{C_4^1} + \frac{1}{2} \times \frac{UVCD_{\lambda=2}(tc_2,S)}{C_4^2} = \frac{1}{2} \times \frac{3}{4} + \frac{1}{2} \times \frac{6}{6} = 0.375 + 0.5 = 0.875$; while the *WASD* of tc_9 against S is $WASD(tc_9, S) = \frac{1}{2} \times \frac{UVCD_{\lambda=1}(tc_9,S)}{C_4^1} + \frac{1}{2} \times \frac{UVCD_{\lambda=2}(tc_9,S)}{C_4^2} = \frac{1}{2} \times \frac{4}{4} + \frac{1}{2} \times \frac{6}{6} = 0.5 + 0.5 = 1.0$. In this case, it can be concluded that the test case tc_9 would be a better next test case in S than tc_2.

4.3. Algorithm

In this section, we introduce a new heuristic algorithm, namely "*aggregate-strength prioritization*" strategy (ASPS), to prioritize interaction test suites using the *WASD* to evaluate combinatorial test cases.

Algorithm 1 Aggregate-strength prioritization strategy (ASPS)

Input: Covering array $CA(N; \tau, k, |V_1||V_2|\cdots|V_k|)$, denoted as T_τ
Output: Interaction test sequence S

```
S ← ⟨ ⟩;
while (|S| ≠ N)
    best_distance ← −1;
    equalSet ← { };
    for (each element e ∈ T_τ)
        Calculate distance ← WASD(e, S);
        if (distance > best_distance)
            equalSet ← { };
            best_distance ← distance;
            best_data ← e;
        else if (distance == best_distance)
            equalSet ← equalSet ⋃ {e};
        end if
    end for
    best_data ← random(equalSet);
    /* Randomly choose an element from equalSet */
    T_τ ← T_τ \ {best_data};
    S ← S ≻ ⟨best_data⟩;
end while
return S.
```

Given a covering array $T = CA(N; \tau, k, |V_1||V_2|\cdots|V_k|)$, the element e' is selected from T as the next test element in an interaction test sequence S when using the *WASD*, such that:

$$(\forall e)\ (e \in T)\ (e \ne e')\ [WASD(e', S) \ge WASD(e, S)]. \quad (8)$$

This process is repeated until all candidates have been selected.

The ASPS algorithm is described in Algorithm 1. In some cases, there may be more than one best test element, indicating that they have the same maximal *WASD* value. In such a situation, a best element is randomly selected.

For ease of description, we use a term T_τ to represent a $CA(N; \tau, k, |V_1||V_2|\cdots|V_k|)$, with the strength τ used in the *WASD* and Algorithm 1 often being provided by a CA rather than being assigned in advance.

4.4. Properties

Consider a T_τ and a pre-selected interaction test sequence $S \subseteq T_\tau$ prioritized by the ASPS algorithm, some properties of ASPS are discussed as follows.

Theorem 1. *Once S covers all possible $(\tau - 1)$-wise value combinations where $2 \le \tau \le k$, the ASPS algorithm has the same mechanism as the "fixed-strength prioritization strategy".*

Proof. When S covers all possible $(\tau - 1)$-wise value combinations, it can be noted that $CombSet_{\tau-1}(S) = CombSet_{\tau-1}(T_\tau)$, which means that

$$(\forall tc)\ (tc \in (T_\tau \setminus S))\ [UVCD_{\tau-1}(tc, S) = 0]. \quad (9)$$

Since

$$\begin{aligned} & CombSet_{\tau-1}(S) = CombSet_{\tau-1}(T_\tau) \\ \Rightarrow\ & CombSet_\lambda(CombSet_{\tau-1}(S)) = CombSet_\lambda(CombSet_{\tau-1}(T_\tau)) \\ \Rightarrow\ & CombSet_\lambda(S) = CombSet_\lambda(T_\tau), \end{aligned} \quad (10)$$

where $1 \le \lambda \le \tau - 1$. Therefore, we can obtain that

$$(\forall tc)\ (tc \in (T_\tau \setminus S))\ [UVCD_\lambda(tc, S) = 0]. \quad (11)$$

where $\lambda = 1, 2, \cdots, \tau - 1$. As a consequence, $WASD(tc, S) = \omega_\tau \times \frac{UVCD_\tau(tc,\ S)}{C_k^\tau}$. Since ω_τ is a constant parameter, $WASD(tc,\ S)$ is only related to $UVCD_\tau(tc, S)$. Consequently, the ASPS algorithm only uses τ-wise interaction coverage to select the next test case, which means that it is "*fixed-strength prioritization*". In summary, once S covers all possible $(\tau - 1)$-wise value combinations, the ASPS algorithm becomes the same as "*fixed-strength prioritization*". □

4.5. Complexity analysis

In this section, we briefly analyze the complexity of the ASPS algorithm (Algorithm 1). Given a $CA(N; \tau, k, |V_1||V_2|\cdots|V_k|)$, denoted as T_τ, we define $\delta = \max_{1 \le i \le k}\{|V_i|\}$.

We first analyze the time complexity of selecting the i-th ($i = 1, 2, \cdots, N$) combinatorial test case, which depends on two factors: (1) the number of candidates required for the calculation of *WASD*; and (2) the time complexity of calculating *WASD* for each candidate.

For (1), $(N-i)+1$ test cases are required to compute *WASD*. For (2), according to C_k^l l-wise parameter combinations where $1 \leq l \leq \tau$, we divide all l-wise value combinations that are derived from a $TP(k, |V_1||V_2|\cdots|V_k|, \mathscr{D})$ into C_k^l sets that form

$$\Pi_l = \{\pi_l | \pi_l = \{(v_{i_1}, v_{i_2}, \cdots, v_{i_l}) | v_{i_j} \in V_{i_j}, j = 1, 2, \cdots, l\}, \\ 1 \leq i_1 < i_2 < \cdots < i_l \leq k\}. \quad (12)$$

Consequently, when using a binary search, the order of time complexity of (2) is $O(\sum_{l=1}^{\tau} \sum_{\pi_l \in \Pi_l} \log(|\pi_l \setminus CombSet_l(T)|))$, which is equal to $O(\sum_{l=1}^{\tau} \sum_{\pi_l \in \Pi_l} \log(|\pi_l|))$.

Therefore, the order of time complexity of algorithm ASPS can be described as follows:

$$\begin{aligned} O(ASPS) &= O((\sum_{i=1}^{N}(N-i+1)) \times (\sum_{l=1}^{\tau}\sum_{\pi_l \in \Pi_l} \log(|\pi_l|))) \\ &< O((\sum_{i=1}^{N}(N-i+1)) \times \sum_{l=1}^{\tau}(C_k^l \times \log(\delta^l))) \\ &= O(\frac{N^2+N}{2} \times \sum_{l=1}^{\tau}(C_k^l \times \log(\delta^l))). \end{aligned} \quad (13)$$

There exists an integer η ($1 \leq \eta \leq \tau$) such that[2]:

$$(\forall l)\ (1 \leq l \leq \tau)\ (\eta \neq l)\ [(C_k^\eta \times \log(\delta^\eta)) \geq (C_k^l \times \log(\delta^l))]. \quad (14)$$

As a consequence,

$$\begin{aligned} O(ASPS) &< O(\frac{N^2+N}{2} \times \sum_{l=1}^{\tau}(C_k^\eta \times \log(\delta^\eta))) \\ &= O(\frac{N^2+N}{2} \times \tau \times C_k^\eta \times \log(\delta^\eta)) \end{aligned} \quad (15)$$

Therefore, we can conclude that the order of time complexity of algorithm ASPS is $O(N^2 \times \tau \times C_k^\eta \times \log(\delta^\eta))$.

The value of τ is usually assigned in the range from 2 to 6 [23, 24], therefore, $1 \leq \tau < \lceil \frac{k}{2} \rceil$, in general. If $1 \leq \tau < \lceil \frac{k}{2} \rceil$, $\eta = \tau$, then the order of time complexity of algorithm ASPS is $O(\tau \times N^2 \times C_k^\tau \times \log(\delta^\tau))$. Since counting the number of value combinations at different strengths can be implemented in parallel, the order of time complexity of the ASPS algorithm can be reduced to $O(N^2 \times C_k^\tau \times \log(\delta^\tau))$. As discussed in [40], the order of time complexity of the ICBP algorithm (an implementation of "*fixed-strength prioritization*") is $O(N^2 \times C_k^\tau \times \log(\delta^\tau))$. The order of time complexity of the inCTPri algorithm (another "*fixed-strength prioritization*" implementation) also equals to $O(N^2 \times C_k^\tau \times \log(\delta^\tau))$ [39]. Therefore, the order of time complexity of the ASPS algorithm is similar to that of both ICBP and inCTPri.

[2] If $1 \leq \tau < \lceil \frac{k}{2} \rceil$, $\eta = \tau$; while if $\lceil \frac{k}{2} \rceil \leq \tau \leq k$, $\eta = \lceil \frac{k}{2} \rceil$.

5. Empirical Study

In this section, some experimental results, including simulations and experiments involving real programs, are presented to analyze the effectiveness of the prioritization of interaction test suites by multiple interaction coverage (multiple strengths). We evaluate interaction test sequences prioritized by algorithm ASPS (denoted `ASPS`) by comparing them with those ordered by four other strategies: (1) test-case-generation prioritization (denoted `Original`), which is an interaction test sequence according to the covering array generation sequence; (2) reverse test-case-generation prioritization (denoted `Reverse`), which is the reversed order of the `Original` interaction test sequence; (3) random test sequence, whose ordering is prioritized in a random manner (denoted `Random`); and (4) two implementations of "*fixed-strength prioritization*" – the ICBP algorithm (denoted `ICBP`) [1, 2, 4–7, 18, 30–33, 37, 40]; and the inCTPri algorithm (denoted `inCTPri`) [39].

In Algorithm 1, which uses *WASD*, it is necessary to assign a weight for each interaction coverage (Eq. (7)). The ideal weight assignment is in accordance with actual fault distribution in terms of the FTFI number. However, the actual fault distribution is unknown before testing. In this paper, we focus on three distribution styles: (1) equal weighting distribution – each interaction coverage has the same weight, that is, $\omega_1 = \omega_2 = \cdots = \omega_\tau = \frac{1}{\tau}$; (2) random weighting distribution – the weight of each interaction coverage is randomly distributed; and (3) empirical FTFI percentage weighting distribution based on previous investigations [23, 24]: for example, in [24], several software projects were studied and the interaction faults reported to have 29% to 82% faults as 1-wise faults, 6% to 47% of faults as 2-wise faults, 2% to 19% as 3-wise faults, 1% to 7% of faults as 4-wise faults, and even fewer faults beyond 4-wise interactions. Consequently, we arranged the weights as follows: $\omega_1 = \omega$, $\omega_{i+1} = \frac{1}{2}\omega_i$, where $i = 1, 2, \cdots, \tau - 1$. For example, if $\tau = 2$, we set $\omega_1 = 0.67$ and $\omega_2 = 0.33$; if $\tau = 3$, $\omega_1 = 0.57$, $\omega_2 = 0.29$, and $\omega_3 = 0.14$. For clarity of description, we use $\texttt{ASPS}_e$, $\texttt{ASPS}_r$, and $\texttt{ASPS}_m$ to represent the ASPS algorithm with equal weighting distribution, random weighting distribution, and empirical FTFI percentage weighting distribution, respectively.

The original covering arrays were generated using two popular tools: (1) *Advanced Combinatorial Testing System* (ACTS) [25, 38]; and (2) *Pairwise Independent Combinatorial Testing* (PICT) [10]. Both ACTS and PICT are supported by greedy algorithms, however, they are implemented by different strategies: ACTS is implemented by the *In-Parameter-Order* (IPO) method [38]; while PICT is implemented by the *one-test-at-a-time* approach [3]. We focused on covering arrays with strength $\tau = 2, 3, 4, 5$. We designed simulations and experiments to answer the following research questions:

RQ1: Do prioritized interaction test sequences generated by ASPS methods ($\texttt{ASPS}_e$, $\texttt{ASPS}_r$, and $\texttt{ASPS}_m$) have better performance (e.g. fault detection) when compared with non-prioritized interaction test suites (`Original`)? The answer

Table 3: Sizes of covering arrays for four test profiles.

Test Profile	ACTS				PICT			
	$\tau = 2$	$\tau = 3$	$\tau = 4$	$\tau = 5$	$\tau = 2$	$\tau = 3$	$\tau = 4$	$\tau = 5$
$TP1(6, 5^6, \emptyset)$	25	199	1058	4149	37	215	1072	4295
$TP2(10, 2^3 3^3 4^3 5^1, \emptyset)$	23	103	426	1559	23	109	411	1363
$TP3(7, 2^4 3^1 6^1 16^1, \emptyset)$	96	289	578	1728	96	293	744	1658
$TP4(8, 2^6 9^1 10^1, \emptyset)$	90	180	632	1080	90	192	592	1237

to this question will help us decide whether it would be necessary to prioritize interaction test suites using ASPS methods.

RQ2: Are ASPS methods better than intuitive prioritization strategies such as the reverse prioritization (`Reverse`) and the random prioritization (`Random`)? The answer to this question will tell us whether or not it would be helpful to use ASPS methods rather than reverse or random ordering for prioritizing interaction test suites.

RQ3: Do ASPS methods perform better than "*fixed-strength prioritization*" (`ICBP` and `inCTPri`)? The answer to this question will help us decide whether or not ASPS can be a promising technique for interaction test suite prioritization, especially if it could perform as effectively as the current best prioritization techniques ("*fixed-strength prioritization*").

RQ4: Which weighting distribution is more suitable for the ASPS method: equal weighting distribution, random weighting distribution, or empirical FTFI percentage weighting distribution? The answer to this question will help us decide which weighting distribution to use for the ASPS method.

5.1. Simulation

We ran a simulation to measure how quickly an interaction test sequence could cover value combinations of different strengths. The simulation details are presented in the following.

5.1.1. Setup

We designed four test profiles as four system models with details as shown in Table 3. The first two test profiles were $TP1(6, 5^6, \emptyset)$ and $TP2(10, 2^3 3^3 4^3 5^1, \emptyset)$, both of which have been used in previous studies [40]. The third and fourth test profiles ($TP3(7, 2^4 3^1 6^1 16^1, \emptyset)$ and $TP4(8, 2^6 9^1 10^1, \emptyset)$) have previously been created [32, 33] to model real-world subjects – a module from a lexical analyzer system (flex), and a real configuration model for GNUzip (gzip)[3].

The sizes of the covering arrays generated by ACTS and PICT are given in Table 3, Since randomization is used in some test case prioritization techniques, we ran each test profile 100 times and report the average of the results.

[3]These two models are unconstrained and incomplete.

5.1.2. Metric

The *average percentage of combinatorial coverage* (APCC) metric[4] [40] is used to evaluate the rate of value combinations covered by an interaction test sequence. The APCC values range from 0% to 100%, with higher values meaning better rates of covering value combinations. Let an interaction test sequence $S = \langle s_1, s_2, \cdots, s_N \rangle$ be obtained by prioritizing a $CA(N; \tau, k, |V_1||V_2|\cdots|V_k|)$, that is, T_τ, the formula to calculate APCC at strength λ ($1 \le \lambda \le \tau$) is:

$$APCC_\lambda(S) = \frac{\sum_{i=1}^{N-1} |\bigcup_{j=1}^{i} CombSet_\lambda(s_j)|}{N \times |CombSet_\lambda(T_\tau)|} \times 100\%. \quad (16)$$

Additionally, since we consider $\lambda = 1, 2, \cdots, \tau$ for an interaction test sequence S of a $CA(N; \tau, k, |V_1||V_2|\cdots|V_k|)$, we could obtain τ APCC values (that is, $APCC_1(S)$, $APCC_2(S)$, $\cdots$, $APCC_\tau(S)$). Therefore, in this simulation we also considered the average of the APCC values, which is defined as:

$$Avg.(S) = \frac{1}{\tau} \sum_{\lambda=1}^{\tau} APCC_\lambda(S). \quad (17)$$

5.1.3. Results and Discussion

For covering arrays of strength τ ($2 \le \tau \le 5$) on individual test profiles, we have the following observations based on the data in Tables 4 and 5, which are separated according to the four test profiles.

1) According to the APCC metric, are prioritized interaction test suites by ASPS methods better than non-prioritized interaction test suites? In this part, we analyze the data to answer whether ASPS methods ($\texttt{ASPS}_e$, $\texttt{ASPS}_r$, and $\texttt{ASPS}_m$) are more effective than `Original`.

Since different weighting distributions in ASPS provide different APCC values, we compare the APCC values of each ASPS method with **Original**. In 98.21% (110/112), 86.61% (97/112), and 98.21% (110/112) of cases, interaction test sequences prioritized by $\texttt{ASPS}_e$, $\texttt{ASPS}_r$, and $\texttt{ASPS}_m$,

[4]In [30], Petke *et al.* proposed a similar metric, namely the *average percentage of covering-array coverage* metric (APCC). Both Wang's APCC [40] and Petke's APCC [30] aim at measuring how quickly an interaction test sequence achieves interaction coverage at a given strength. In fact, their APCCs are equivalent: given two interaction test sequences, S_1 and S_2, if one determines that the test sequence S_1 is better than S_2, then the other metric will also have the same determination. The only difference between them is that they use different plot curves to describe the rate of covered value combinations. More specifically, Wang's APCC [40] uses the ladder chart; while Petke's APCC [30] uses the line chart.

Table 4: $APCC_\lambda$ metric (%) for different prioritization techniques for $TP1(6, 5^6, \emptyset)$ and $TP2(10, 2^3 3^3 4^3 5^1, \emptyset)$.

	Method	τ = 2			τ = 3				τ = 4					τ = 5					
		λ=1	λ=2	*Avg.*	λ=1	λ=2	λ=3	*Avg.*	λ=1	λ=2	λ=3	λ=4	*Avg.*	λ=1	λ=2	λ=3	λ=4	λ = 5	*Avg.*
TP1 – ACTS	Original	82.67	**48.00**	*65.34*	93.80	85.11	63.47	*80.79*	94.93	89.61	82.68	63.59	*82.70*	93.69	87.37	80.91	73.18	57.60	*78.55*
	Reverse	82.67	**48.00**	*65.34*	97.91	88.23	55.61	*80.58*	99.50	97.67	88.74	57.17	*85.77*	98.77	94.88	87.27	72.53	49.56	*80.60*
	Random	82.75	**48.00**	*65.38*	97.54	80.63	58.62	*78.93*	99.53	97.69	89.20	59.99	*86.60*	99.88	99.40	97.06	86.49	54.90	*87.55*
	ICBP	82.96	**48.00**	*65.48*	97.71	89.94	**64.31**	*83.99*	99.54	97.88	91.36	**65.39**	*88.54*	99.88	99.42	97.19	87.98	**60.42**	*88.99*
	inCTPri	82.63	**48.00**	*65.32*	98.21	**92.14**	63.59	*84.65*	99.66	**98.62**	92.53	64.88	*88.92*	99.91	**99.66**	98.19	90.67	60.11	*89.71*
	$ASPS_e$	**85.87**	**48.00**	***66.94***	**98.45**	91.67	64.13	*84.75*	**99.71**	98.60	92.40	65.30	*89.00*	**99.93**	99.65	98.18	**90.75**	60.38	***89.78***
	$ASPS_r$	83.27	**48.00**	*65.63*	98.31	90.43	63.22	*83.99*	99.70	98.20	91.09	64.43	*88.35*	99.92	99.55	97.61	88.49	59.93	*89.10*
	$ASPS_m$	**85.87**	**48.00**	***66.94***	**98.45**	92.09	63.88	***84.81***	**99.71**	98.60	**92.56**	65.19	***89.02***	**99.93**	99.65	**98.20**	90.73	60.39	***89.78***
TP1 – PICT	Original	90.63	**60.27**	*75.45*	98.16	92.11	64.40	*84.89*	99.59	97.83	91.41	64.56	*88.35*	99.89	99.40	97.30	88.49	59.27	*88.87*
	Reverse	89.28	56.37	*72.83*	97.85	89.89	59.87	*82.54*	99.59	97.81	89.68	59.60	*86.67*	99.89	99.40	97.22	87.18	54.73	*87.68*
	Random	87.52	56.35	*71.94*	97.70	89.39	60.26	*82.45*	99.53	97.73	89.37	60.32	*86.74*	99.88	99.42	97.17	87.12	55.94	*87.91*
	ICBP	89.95	**60.27**	*75.11*	97.91	91.79	**64.58**	*84.76*	99.55	97.90	91.53	**65.28**	*88.57*	99.88	99.42	97.31	88.59	**60.99**	*89.24*
	inCTPri	89.95	**60.27**	*75.11*	98.34	**92.79**	64.29	*85.14*	99.66	**98.63**	92.66	64.83	*88.95*	99.92	**99.67**	98.25	90.99	60.60	*89.89*
	$ASPS_e$	91.16	60.11	*75.64*	98.56	92.58	**64.58**	*85.24*	**99.72**	98.62	92.53	65.19	*89.02*	**99.93**	**99.67**	98.25	**91.09**	60.90	***89.97***
	$ASPS_r$	90.30	59.44	*74.87*	98.44	91.39	63.75	*84.52*	99.70	98.22	91.26	64.33	*88.38*	**99.93**	99.57	97.70	88.99	60.43	*89.32*
	$ASPS_m$	**91.21**	60.12	***75.67***	**98.57**	92.73	64.47	***85.26***	**99.72**	**98.63**	**92.69**	65.08	***89.03***	**99.93**	**99.67**	**98.26**	91.06	60.90	***89.97***
TP2 – ACTS	Original	86.14	66.55	*76.35*	92.72	85.33	72.17	*83.41*	97.06	88.66	82.99	73.82	*85.63*	95.31	91.08	87.13	83.01	76.11	*86.53*
	Reverse	84.92	60.93	*72.92*	96.69	88.78	69.20	*84.89*	99.14	97.36	90.15	73.14	*89.95*	99.61	98.81	97.19	91.87	76.49	*92.79*
	Random	86.15	62.75	*74.45*	96.69	89.52	70.98	*85.73*	99.19	97.28	91.45	76.11	*91.01*	99.77	99.24	97.54	92.42	79.00	*93.59*
	ICBP	88.60	**67.32**	*77.96*	97.56	91.85	**74.99**	*88.13*	99.36	98.03	93.51	**79.98**	*92.72*	99.81	99.44	98.20	94.25	82.82	*94.90*
	inCTPri	88.60	**67.32**	*77.96*	97.67	**92.23**	74.43	*88.11*	99.45	**98.18**	93.62	79.37	*92.66*	99.85	**99.51**	98.30	94.27	82.18	*94.82*
	$ASPS_e$	88.89	67.30	*78.10*	97.70	91.86	74.95	*88.17*	99.46	98.04	93.52	79.97	*92.75*	**99.86**	99.48	98.24	94.29	**82.83**	*94.94*
	$ASPS_r$	88.35	66.40	*77.38*	97.47	91.24	73.93	*87.54*	99.40	97.78	92.83	78.98	*92.25*	99.84	99.38	97.95	93.63	81.81	*94.52*
	$ASPS_m$	**89.31**	67.24	***78.28***	**97.75**	92.14	74.83	***88.24***	**99.47**	98.14	**93.65**	79.89	***92.79***	**99.86**	99.50	**98.31**	**94.38**	82.77	***94.96***
TP2 – PICT	Original	88.18	66.51	*77.35*	97.56	92.21	76.23	*88.67*	99.08	97.44	92.55	78.56	*91.91*	99.78	99.27	97.65	92.88	79.83	*93.88*
	Reverse	86.68	63.29	*74.99*	97.36	90.41	71.47	*86.41*	99.06	97.25	91.34	74.83	*90.62*	99.72	99.13	97.36	91.81	76.71	*92.94*
	Random	86.16	63.23	*74.70*	96.90	90.05	72.10	*86.35*	99.15	97.18	91.22	75.45	*90.75*	99.75	99.15	97.21	91.49	76.86	*92.89*
	ICBP	88.64	**66.82**	*77.73*	97.70	92.32	**76.10**	*88.71*	99.34	97.95	93.26	**79.17**	*92.43*	99.78	99.36	97.93	93.41	**80.39**	*94.17*
	inCTPri	88.64	**66.82**	*77.73*	97.81	**92.68**	75.60	*88.70*	99.43	**98.11**	93.38	78.53	*92.36*	**99.83**	**99.44**	98.05	93.44	79.66	*94.08*
	$ASPS_e$	88.85	66.78	*77.82*	97.85	92.32	76.08	*88.75*	99.44	97.96	93.31	79.15	*92.47*	**99.83**	99.40	97.97	93.45	80.36	*94.20*
	$ASPS_r$	88.17	65.98	*77.07*	97.63	91.68	75.03	*88.11*	99.37	97.71	92.57	78.10	*91.94*	99.81	99.30	97.66	92.72	79.31	*93.76*
	$ASPS_m$	**89.18**	66.70	***77.94***	**97.88**	92.60	75.98	***88.82***	**99.45**	98.08	**93.43**	79.08	***92.51***	**99.83**	99.43	**98.06**	**93.56**	80.28	***94.23***

Table 5: $APCC_\lambda$ metric (%) for different prioritization techniques for $TP3(7, 2^4 3^1 6^1 16^1, \emptyset)$ and $TP4(8, 2^6 9^1 10^1, \emptyset)$.

	Method	τ = 2			τ = 3				τ = 4					τ = 5					
		λ=1	λ=2	*Avg.*	λ=1	λ=2	λ=3	*Avg.*	λ=1	λ=2	λ=3	λ=4	*Avg.*	λ=1	λ=2	λ=3	λ=4	λ = 5	*Avg.*
TP3 – ACTS	Original	75.54	63.40	*69.47*	76.46	65.40	58.65	*66.84*	76.65	65.68	59.50	55.18	*64.25*	84.45	77.11	73.02	70.41	66.21	*74.24*
	Reverse	75.28	59.76	*67.52*	79.01	66.45	56.49	*67.32*	79.18	67.94	60.36	52.14	*64.91*	95.56	92.86	90.01	81.37	63.11	*84.58*
	Random	91.07	69.82	*80.45*	96.85	87.64	68.32	*84.27*	98.38	93.26	81.98	61.31	*83.73*	99.48	97.61	92.84	82.89	65.39	*87.64*
	ICBP	93.98	**75.77**	*84.88*	97.79	90.91	73.82	*87.51*	98.66	94.52	84.12	**64.78**	*85.52*	99.51	98.10	94.75	87.22	**71.29**	*90.17*
	inCTPri	93.98	**75.77**	*84.88*	98.09	**92.12**	72.57	*87.59*	99.05	**96.11**	85.64	62.61	*85.85*	**99.69**	**98.71**	95.56	87.28	70.00	*90.25*
	$ASPS_e$	94.14	75.71	*84.93*	98.04	91.36	**73.85**	*87.75*	99.06	95.43	84.57	64.75	*85.95*	**99.69**	98.59	95.28	87.67	**71.29**	*90.50*
	$ASPS_r$	94.13	75.38	*84.75*	97.82	90.69	73.26	*87.26*	98.82	94.58	84.02	64.09	*85.38*	99.60	98.14	94.47	86.45	70.48	*89.83*
	$ASPS_m$	**94.47**	75.69	***85.08***	**98.17**	92.09	73.60	***87.95***	**99.08**	96.04	**85.84**	64.33	***86.32***	**99.69**	**98.71**	**95.68**	**87.89**	71.07	***90.61***
TP3 – PICT	Original	92.58	74.52	*83.55*	97.25	88.47	72.28	*86.00*	98.70	94.74	86.57	70.84	*87.71*	99.47	97.50	93.13	84.60	68.28	*88.60*
	Reverse	91.13	70.56	*80.85*	96.94	87.74	69.20	*84.62*	98.77	94.51	85.63	67.68	*86.64*	99.40	97.54	92.80	82.72	64.60	*87.41*
	Random	91.12	71.04	*81.08*	96.89	87.81	69.80	*84.83*	98.72	94.62	85.05	67.62	*86.50*	99.41	97.50	92.55	82.31	64.80	*87.31*
	ICBP	94.17	**76.27**	*85.22*	97.90	91.36	**75.02**	*88.09*	99.05	96.24	88.87	**72.79**	*89.24*	99.51	98.01	94.55	86.65	**70.17**	*89.78*
	inCTPri	94.17	**76.27**	*85.22*	98.14	**92.19**	73.96	*88.10*	99.26	**97.00**	89.31	71.63	*89.30*	99.67	**98.66**	**95.77**	86.79	68.95	*89.97*
	$ASPS_e$	94.37	76.26	*85.32*	98.11	91.71	74.98	*88.27*	99.27	96.59	89.15	72.76	*89.44*	**99.68**	98.49	95.05	87.14	70.16	*90.10*
	$ASPS_r$	94.16	75.93	*85.05*	97.85	90.89	74.48	*87.73*	99.10	95.98	88.24	71.99	*88.83*	99.58	98.07	94.17	85.76	69.30	*89.38*
	$ASPS_m$	**94.48**	76.24	***85.36***	**98.19**	92.18	74.88	***88.42***	**99.29**	96.98	**89.52**	72.60	***89.60***	**99.68**	98.65	95.48	**87.36**	69.95	***90.22***
TP4 – ACTS	Original	82.87	69.62	*76.25*	83.53	71.77	62.21	*72.50*	90.80	84.17	79.60	72.11	*81.67*	89.34	81.61	76.56	72.95	66.28	*77.35*
	Reverse	82.58	68.03	*75.30*	83.53	71.41	54.73	*69.89*	98.75	95.38	87.43	70.24	*87.95*	97.31	94.47	89.94	80.65	61.19	*84.71*
	Random	93.26	76.03	*84.83*	96.38	85.75	64.66	*82.26*	98.96	95.14	86.82	71.32	*88.06*	99.38	96.97	91.71	81.67	64.62	*86.87*
	ICBP	95.58	**79.94**	*87.76*	97.32	84.49	69.40	*83.74*	99.26	96.50	90.60	76.65	*90.75*	99.52	97.55	93.29	85.98	**69.96**	*89.26*
	inCTPri	95.58	**79.94**	*87.76*	97.69	**89.75**	66.55	*84.66*	99.38	**97.15**	90.57	75.32	*90.61*	99.64	**98.32**	94.69	85.78	68.22	*89.33*
	$ASPS_e$	95.72	79.89	*87.81*	97.86	87.15	**69.42**	*84.81*	99.38	96.70	90.77	**76.69**	*90.89*	**99.65**	98.00	93.93	86.23	69.63	*89.49*
	$ASPS_r$	95.58	79.70	*87.64*	97.50	87.01	68.89	*84.47*	99.25	96.38	89.99	75.94	*90.39*	99.56	97.65	93.27	84.83	68.97	*88.86*
	$ASPS_m$	**95.76**	79.89	***87.83***	**97.88**	89.14	68.69	***85.24***	**99.40**	97.11	**91.07**	76.57	***91.04***	**99.65**	98.29	**94.84**	**86.55**	69.27	***89.72***
TP4 – PICT	Original	93.94	78.62	*86.28*	97.08	88.32	71.00	*85.47*	98.91	95.37	88.47	74.28	*89.26*	99.54	97.71	93.71	86.16	71.56	*89.74*
	Reverse	94.05	75.46	*84.76*	96.94	87.83	69.59	*84.79*	98.85	95.06	87.30	71.18	*88.10*	99.48	97.46	93.12	84.52	68.16	*88.55*
	Random	93.17	75.82	*84.50*	96.71	86.45	66.87	*83.34*	98.90	94.84	86.18	70.87	*87.70*	99.44	97.45	92.77	83.64	68.08	*88.28*
	ICBP	95.51	**79.94**	*87.73*	97.58	88.88	72.20	*86.22*	99.21	96.40	89.94	75.43	*90.25*	99.57	98.07	94.67	87.72	**72.95**	*90.60*
	inCTPri	95.51	**79.94**	*87.73*	97.94	**90.00**	70.78	*86.24*	99.35	**96.98**	89.98	74.49	*90.20*	**99.69**	**98.53**	95.32	87.52	71.81	*90.57*
	$ASPS_e$	95.73	79.84	***87.79***	97.90	89.76	**72.22**	*86.63*	99.35	96.52	90.08	**75.45**	*90.35*	**99.69**	98.31	94.95	87.91	72.94	*90.76*
	$ASPS_r$	95.60	79.61	*87.61*	97.71	88.89	71.75	*86.12*	99.22	96.12	89.22	74.66	*89.80*	99.62	97.99	94.25	86.72	71.94	*90.10*
	$ASPS_m$	**95.76**	79.82	***87.79***	**98.02**	89.99	72.10	***86.70***	**99.36**	96.95	**90.44**	75.39	***90.54***	**99.69**	**98.53**	**95.51**	**88.12**	72.71	***90.91***

respectively, have greater APCC values than Original test sequences. Additionally, the average APCC values of $\texttt{ASPS}_e$, $\texttt{ASPS}_r$, and $\texttt{ASPS}_m$ are higher than those of Original, in 100.00% (32/32), 84.38% (27/32), and 100.00% (32/32) of cases, respectively.

As shown in Tables 4 and 5, it can be noted that different non-prioritized interaction test suites generated by different tools have different performances.

For example, consider $TP3(7, 2^4 3^1 6^1 16^1, \emptyset)$ at strength $\tau = 2$: when non-prioritized covering arrays are constructed using ACTS, the difference between $\texttt{ASPS}_e$ and Original is 18.60% for $\lambda = 1$, and 32.31% for $\lambda = 2$. However, when using PICT, the difference is 1.79% for $\lambda = 1$, and 1.74% for $\lambda = 2$. The main reason for this is related to the different mechanisms used in the ACTS and PICT tools [10, 25, 38]. Specifically, without loss of generality, consider a test profile $TP(k, |V_1||V_2|\cdots|V_k|, \emptyset)$ with $|V_1| \geq |V_2| \geq \cdots \geq |V_k|$. When generating a τ-wise ($1 \leq \tau \leq k$) covering array, the ACTS algorithm first uses *horizontal growth* [25, 38] to construct a τ-wise test set for the first τ parameters, which implies that it needs at least $(1 + (|V_1| - 1) \times \prod_{i=2}^{\tau} |V_i|)$ test cases to cover all possible 1-wise value combinations. However, the PICT algorithm chooses each next test case such that it covers the largest number of τ-wise value combinations that have not yet been covered – a mechanism similar to that of ICBP.

In conclusion, the simulation indicates that the ASPS techniques do outperform Original in terms of the rate of covering value combinations, regardless of which construction tools are used (ACTS or PICT).

2) Do ASPS methods have better APCC values than reverse or random ordering? In this part, we attempt to determine whether or not ASPS methods are more effective than two widely-used prioritization methods, Reverse and Random.

In all cases, each ASPS method (regardless of $\texttt{ASPS}_e$, $\texttt{ASPS}_r$, and $\texttt{ASPS}_m$) has higher APCC values than Reverse, and hence achieves higher average APCC values. Additionally, the performance of Reverse is correlated with the non-prioritized interaction test suite (that is, the ACTS and PICT tools).

Compared with Random, the ASPS methods have higher APCC values in all cases, irrespective of the strength $\lambda = 1, 2, \cdots, \tau$ and interaction test suite construction tool (ACTS or PICT). As a result, the ASPS methods have better performance according to the average APCC values at different strengths.

In conclusion, in all cases our ASPS methods (regardless of the weighting distributions) do perform better than both the Reverse and Random prioritization strategies, according to the APCC values.

3) Are ASPS methods better than "fixed-strength prioritization"? In this part, we would like to determine whether or not ASPS methods perform better than two implementations of "*fixed-strength prioritization*", ICBP and inCTPri.

Compared with ICBP, according to APCC values, $\texttt{ASPS}_e$, $\texttt{ASPS}_r$ and $\texttt{ASPS}_m$ perform better in 79.46% (89/112), 36.61% (41/112), and 72.32% (81/112) of cases, respectively. Furthermore, according to the average of APCC values (*Avg.*), $\texttt{ASPS}_e$, $\texttt{ASPS}_r$, and $\texttt{ASPS}_m$ have better performances than ICBP in 100.00% (32/32), 18.75% (6/32), and 100.00% (32/32) of cases.

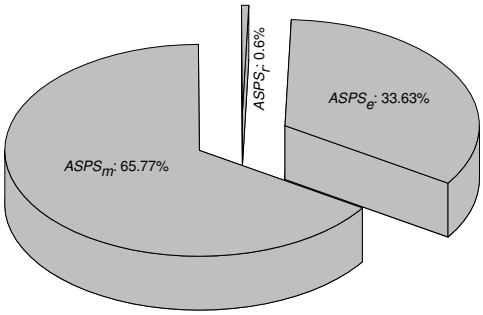

Figure 1: Comparison of $\texttt{ASPS}_e$, $\texttt{ASPS}_r$, and $\texttt{ASPS}_m$ according to APCC.

Similarly, compared with inCTPri, $\texttt{ASPS}_e$, $\texttt{ASPS}_r$, and $\texttt{ASPS}_m$ have higher APCC values in 58.93% (66/112), 21.43% (24/112), and 75.00% (84/112) of cases, respectively. Moreover, according to the average of APCC values, $\texttt{ASPS}_e$, $\texttt{ASPS}_r$, and $\texttt{ASPS}_m$ outperform inCTPri in 100.00% (32/32), 3.13% (1/32), and 100.00% (32/32) of cases.

In conclusion, the simulation results indicate that apart from $\texttt{ASPS}_r$, in (58.93% ~ 79.46%) of cases our ASPS methods ($\texttt{ASPS}_e$ and $\texttt{ASPS}_m$) do perform better than ICBP and inCTPri, and also have higher averages of APCC values in all cases. Consequently, we conclude that our ASPS methods (except for $\texttt{ASPS}_r$) do have better performance than "*fixed-strength prioritization*".

4) Among the three weighting distributions, which weighting distribution is used for the ASPS method? In this part, we are interested in which weighting distribution is more suitable for the ASPS method. As discussed before, there are three distributions used for the ASPS methods: equal, random, and empirical FTFI percentage weighting distributions.

As discussed in the last part, among the three weight distributions for ASPS, $\texttt{ASPS}_r$ has the lowest $APCC_\lambda$ performance, irrespective of which λ value is used ($1 \leq \lambda \leq \tau$). Additionally, when λ is high, $\texttt{ASPS}_e$ performs better than $\texttt{ASPS}_m$, otherwise it performs worse. According to the average APCC values, however, $\texttt{ASPS}_m$ performs best, followed by $\texttt{ASPS}_e$; while $\texttt{ASPS}_r$ has the worst performance. Figure 1 shows the comparison of $\texttt{ASPS}_e$, $\texttt{ASPS}_r$, and $\texttt{ASPS}_m$ according to the APCC metric. From this figure, it can be observed that in 65.77% (73.67/112) of cases $\texttt{ASPS}_m$ has the highest APCC values; in 33.63% (37.67/112) of cases, $\texttt{ASPS}_e$ does; and only in 0.60% (0.67/112) of cases is the highest values for $\texttt{ASPS}_r$[5]. In other words, among three weighting distributions, empirical FTFI percentage weighting distribution would be the best choice, followed by the equal weighting distribution, according to the APCC metric.

[5] If there exist two or three ASPS strategies that have the same APCC value, each strategy is assigned by 1/2 or 1/3.

Table 6: Subject programs.

Subject Program	Test Profile	*#uLOC*	*#Seeded Faults*	*#Detectable Faults*	*#Used Faults*
count	$TP(6, 2^1 3^5, \emptyset)$	42	15	12	12
nametbl	$TP(5, 2^1 3^2 5^2, \emptyset)$	329	51	44	43
flex	$TP(9, 2^6 3^2 5^1, \mathscr{D})$,	9,581 ~ 11,470	81	50	34
grep	$TP(9, 2^1 3^3 4^2 5^1 6^1 8^1, \mathscr{D})$	9,493 ~ 10,173	57	12	10

In conclusion, the empirical FTFI percentage weighting distribution appears to be more suitable than the other weighting distributions for the ASPS method ($\texttt{ASPS}_m$).

5) Conclusion: Based on the above discussions, we find that given a covering array of strength τ, the ASPS strategies behave better than the `Original`, `Reverse`, and `Random` strategies (in 86.61% ~ 100.00% of cases), and apart from $\texttt{ASPS}_r$ perform better than "*fixed-strength prioritization*" including `ICBP` and `inCTPri` implementations (in 58.93% ~ 79.46% of cases). Additionally, among the three weighting distributions, the empirical FTFI percentage is most suitable for use with the ASPS method, followed by the equal weighting distribution.

5.2. Experiments

An experimental study was also conducted to evaluate the ASPS techniques, the goal of which was to compare the fault detection rates of the $\texttt{ASPS}_e$, $\texttt{ASPS}_r$, and $\texttt{ASPS}_m$ techniques against those of other interaction test suite prioritization techniques, such as `Original`, `Reverse`, `Random`, `ICBP`, and `inCTPri`. In actual testing conditions, testing resources may be limited, and hence only part of an interaction test suite (or an interaction test sequence) may be executed. As a consequence, in this study we focused on different budgets by following the practice adopted in previous prioritization studies [30] of considering different percentages (p) of each interaction test sequence, e.g. p = 5%, 10%, 25%, 50%, 75%, and 100% of each interaction test sequence being executed.

5.2.1. Setup

For the study, we used two small-sized faulty C programs (count and nametbl)[6], which had previously been used in research comparing defect revealing mechanisms [27], evaluating different combination strategies for test case selection [17], and fault diagnosis [16, 45]. Since these two programs are small, we also used another two medium-sized UNIX utility programs[7], flex and grep, with real and seeded faults from the Software-artifact Infrastructure Repository (SIR) [11]. These two programs had also been widely used in prioritization research [22, 30, 32]. To determine the correctness of an executing test case, we created a fault-free version of each program (i.e. an oracle) by analyzing the corresponding fault description. These subject programs are described in Table 6, in which the *Test Profile* is the test profile of each subject program[8]; the *#uLOC*[9] gives the number of lines of executable code in each program; the *#Seeded Faults*[10] is the number of faults seeded in each program; the *#Detectable Faults* is the number of faults that could be detected from the accompanying test profiles, which are not guaranteed to be able to detect all faults; and the *#Used Faults* is the number of faults used in the experiment by removing some faults that could be triggered by nearly every combinatorial test case in the test suite, that is, faults could be removed such that they are identified by more than (78.00% ~ 100.00%) test cases in the test suite.

Similar to the simulation (Section 5.1), we also used ACTS and PICT to generate original interaction test sequences for each subject program. Additionally, we focused on covering arrays with strength $\tau = 2, 3, 4, 5$. Table 7 gives the sizes of the original interaction test sequences obtained by ACTS and PICT. Because of the randomization in some of the prioritization techniques, we ran the experiment 100 times for each subject program and report the average.

5.2.2. Metric

The APFD metric [13] is a popular measure for evaluating fault detection rates of interaction test sequence. In this study, only part of the interaction test sequences could be run, and some faults might not have been triggered by a particular interaction test sequence. As discussed before, however,

[6] From `http://www.maultech.com/chrislott/work/exp/`

[7] From `http://sir.unl.edu`

[8] The test profiles of two medium-sized programs, flex and grep, are from Petke *et al.* [30].

[9] We used the line count tool named `cloc`, downloaded from `http://cloc.sourceforge.net`, to count the number of code lines.

[10] Similar to [30], in this study we only used the faults provided with each of subject programs, in order to avoid experiment bias and ensure repeatability.

Table 7: Sizes of original interaction test sequences for each subject program.

Subject Program	ACTS				PICT			
	$\tau = 2$	$\tau = 3$	$\tau = 4$	$\tau = 5$	$\tau = 2$	$\tau = 3$	$\tau = 4$	$\tau = 5$
count	15	41	108	243	14	43	116	259
nametbl	25	82	225	450	25	78	226	450
flex	27	66	130	238	26	65	129	219
grep	46	153	298	436	47	150	296	436

APFD has two requirements which may cause APFD to fail. Consequently, it was not possible to use APFD to investigate the fault detection rates of the different prioritization strategies, and so we used an enhanced version of APFD, NAPFD [32], as an alternative evaluation metric.

5.2.3. Results and Discussions

The experimental results from executing all prioritization techniques to test count, nametbl, flex, and grep are summarized in Tables 8 ~ 11, based on which we can have the following observations. It should be noted that the data in bold in the tables is the largest in each sub-column.

1) Does the ASPS method have faster fault detection rates than the `Original` *method?* In this part, we analyze the experimental data to answer the research question of whether or not the ASPS method is better than `Original` according to fault detection rates.

As shown in Tables 8 ~ 11, in 96.84% (184/190), 97.37% (185/190), and 96.84% (184/190) of cases, $\texttt{ASPS}_e$, $\texttt{ASPS}_r$, and $\texttt{ASPS}_m$, respectively, obtain interaction test sequences with higher fault detection rates than `Original`. The fault detection improvement of ASPS over `Original` for ACTS is larger than that for PICT: as was the case in the simulation, the main reason for this is the different methods used to construct ACTS and PICT interaction test suites.

Additionally, as the proportion of the interaction test sequence executed (p) increases, the NAPFD improvement of ASPS over `Original` generally becomes smaller. For example, consider subject program nametbl for ACTS with $\tau = 5$, when $p = 5\%$, 10%, 25%, 50%, 75%, and 100%, the corresponding NAPFD improvements of $\texttt{ASPS}_e$ over `Original` are 58.92%, 42.87%, 27.78%, 14.54%, 9.71%, and 7.27%, respectively.

In conclusion, in (97.37% ~ 96.84%) of cases, the ASPS method has higher rates of fault detection compared with `Original`. Furthermore, the ASPS method favors the cases where smaller percentages of interaction test sequence are executed, compared with `Original`.

Table 8: The NAPFD metric (%) for different prioritization techniques for subject program count when executing the percentage of interaction test sequence.

Method	Strength	p of ACTS Interaction Test Sequence Executed						p of PICT Interaction Test Sequence Executed					
		5%	10%	25%	50%	75%	100%	5%	10%	25%	50%	75%	100%
`Original`	$\tau = 2$	–	0	0	25.00	48.11	59.72	–	0	33.33	69.05	78.33	84.52
`Reverse`		–	0	30.56	62.50	73.11	78.06	–	**25.00**	43.06	51.79	60.00	71.43
`Random`		–	15.38	37.75	60.08	70.76	76.31	–	17.71	44.18	68.19	76.71	83.21
`ICBP`		–	16.29	40.56	61.98	72.05	77.27	–	17.96	46.67	72.69	80.82	86.30
`inCTPri`		–	16.29	40.56	61.98	72.05	77.27	–	17.96	46.67	72.69	80.82	86.30
$\texttt{ASPS}_e$		–	15.21	38.81	62.32	72.09	77.29	–	19.04	**50.68**	**75.45**	**82.82**	**87.73**
$\texttt{ASPS}_r$		–	**17.83**	**42.69**	**63.86**	**73.18**	**78.10**	–	17.08	47.72	72.81	80.62	86.15
$\texttt{ASPS}_m$		–	13.58	39.10	62.90	72.32	77.45	–	17.13	47.24	73.99	81.79	86.99
`Original`	$\tau = 3$	0	0	1.25	31.04	53.33	65.85	12.50	34.38	65.00	83.33	89.06	91.86
`Reverse`		**39.58**	48.96	58.75	74.58	83.06	87.60	0	21.88	55.83	78.97	86.20	89.73
`Random`		30.40	47.89	71.23	84.03	89.08	91.98	29.96	46.75	71.65	85.79	90.67	93.06
`ICBP`		29.06	48.25	73.25	86.03	90.64	93.15	27.25	48.43	75.91	88.49	92.45	94.38
`inCTPri`		31.71	51.96	75.85	**87.05**	**91.34**	**93.66**	32.98	54.44	78.45	89.68	93.23	94.96
$\texttt{ASPS}_e$		32.63	54.40	75.98	86.83	91.16	93.53	30.92	54.42	**78.92**	**89.90**	**93.37**	**95.07**
$\texttt{ASPS}_r$		33.21	53.60	75.45	86.68	91.09	93.48	**34.42**	53.15	77.06	88.74	92.61	94.50
$\texttt{ASPS}_m$		35.00	**55.78**	**76.39**	86.94	91.23	93.58	30.90	**54.75**	78.74	89.85	93.34	95.04
`Original`	$\tau = 4$	0	0	0	28.86	51.03	63.27	60.83	80.30	92.53	96.26	97.51	98.13
`Reverse`		**65.00**	**82.50**	**93.52**	**96.76**	**97.84**	**98.38**	**74.17**	**87.12**	**95.11**	**97.56**	**98.37**	**98.78**
`Random`		57.02	73.14	89.02	94.48	96.32	97.24	52.12	72.47	88.98	94.49	96.33	97.25
`ICBP`		56.04	74.30	90.15	95.08	96.72	97.54	52.76	74.07	89.96	94.98	96.65	97.49
`inCTPri`		60.20	77.81	91.52	95.76	97.17	97.88	59.88	79.42	92.04	96.02	97.35	98.01
$\texttt{ASPS}_e$		61.80	78.25	91.78	95.89	97.26	97.95	60.28	79.11	91.94	95.97	97.31	97.99
$\texttt{ASPS}_r$		60.67	77.70	91.40	95.69	97.13	97.85	60.85	78.61	91.41	95.71	97.14	97.85
$\texttt{ASPS}_m$		61.63	78.34	91.64	95.82	97.21	97.91	60.39	79.32	91.94	95.97	97.31	97.98
`Original`	$\tau = 5$	0	0	0	16.53	35.92	51.75	70.14	85.67	94.40	97.22	98.15	98.62
`Reverse`		78.47	82.81	88.13	92.63	95.10	96.33	80.56	90.67	96.35	98.19	98.80	99.10
`Random`		74.65	86.21	94.37	97.21	98.14	98.61	74.84	86.84	94.77	97.41	98.28	98.71
`ICBP`		76.34	87.46	94.97	97.50	98.34	98.76	76.84	88.17	95.34	97.69	98.46	98.85
`inCTPri`		80.51	90.08	96.03	98.03	98.69	99.02	81.19	90.77	96.40	98.21	98.81	99.11
$\texttt{ASPS}_e$		80.76	90.27	96.11	98.07	98.72	99.04	80.39	90.41	96.25	98.14	98.76	99.07
$\texttt{ASPS}_r$		79.89	89.48	95.79	97.91	98.61	98.96	80.34	90.27	96.20	98.11	98.75	99.06
$\texttt{ASPS}_m$		**81.45**	**90.65**	**96.26**	**98.15**	**98.77**	**99.08**	**81.41**	**90.88**	**96.44**	**98.23**	*98.82*	**99.12**

Table 9: The NAPFD metric (%) for different prioritization techniques for subject program nametbl when executing the percentage of interaction test sequence.

Method	Strength	p of ACTS Interaction Test Sequence Executed						p of PICT Interaction Test Sequence Executed					
		5%	10%	25%	50%	75%	100%	5%	10%	25%	50%	75%	100%
Original	$\tau = 2$	0	15.70	55.04	75.87	83.91	88.42	11.63	27.33	58.72	72.97	81.98	87.02
Reverse		**37.21**	**57.56**	**81.78**	**90.02**	**93.35**	**95.21**	11.63	23.84	61.43	79.36	86.24	90.09
Random		18.22	32.92	62.85	79.31	86.15	90.03	19.13	33.58	61.53	77.54	84.65	88.91
ICBP		19.15	34.49	68.01	83.55	89.04	92.11	18.92	33.98	65.27	80.48	86.90	90.57
inCTPri		19.15	34.49	68.01	83.55	89.04	92.11	18.92	33.98	65.27	80.48	86.90	90.57
$ASPS_e$		19.38	35.81	69.56	84.33	89.55	92.48	19.21	34.52	66.05	81.31	87.50	91.00
$ASPS_r$		18.21	34.21	67.28	83.06	88.71	91.87	**19.23**	34.35	64.69	80.26	86.72	90.44
$ASPS_m$		19.57	35.80	68.70	83.89	89.26	92.27	18.99	**35.07**	**67.22**	**82.28**	**88.14**	**91.46**
Original	$\tau = 3$	27.03	48.55	69.19	84.37	89.50	92.19	40.31	64.12	85.37	92.87	95.21	96.44
Reverse		**72.38**	**82.99**	**91.63**	95.77	**97.16**	**97.89**	18.60	53.16	82.19	91.32	94.17	95.66
Random		53.44	71.53	87.80	94.04	96.00	97.02	43.26	66.39	85.87	93.06	95.34	96.53
ICBP		53.77	72.93	88.91	94.59	96.36	97.29	46.24	69.31	88.11	94.21	96.10	97.10
inCTPri		60.26	78.46	91.32	95.77	**97.16**	97.88	**48.86**	71.82	89.26	94.77	96.48	97.38
$ASPS_e$		60.53	78.59	91.35	**95.78**	**97.16**	**97.89**	47.54	**72.83**	**89.68**	**94.97**	**96.62**	**97.48**
$ASPS_r$		58.32	76.43	90.37	95.30	96.84	97.65	47.21	71.03	88.35	94.25	96.14	97.13
$ASPS_m$		57.75	77.18	90.83	95.53	96.99	97.76	46.28	71.78	89.13	94.70	96.43	97.35
Original	$\tau = 4$	32.03	51.64	70.14	84.35	89.57	92.21	72.09	85.68	94.37	97.21	98.14	98.61
Reverse		81.18	87.47	92.69	95.63	97.09	97.82	74.84	86.63	94.75	97.40	98.26	98.70
Random		77.58	88.29	95.39	97.69	98.46	98.85	78.14	88.57	95.49	97.76	98.50	98.88
ICBP		77.56	88.47	95.47	97.74	98.49	98.87	77.94	88.71	95.56	97.80	98.53	98.90
inCTPri		82.07	90.98	96.46	98.23	98.82	99.12	81.48	90.72	96.35	98.19	98.79	99.10
$ASPS_e$		**83.20**	**91.56**	**96.68**	**98.34**	**98.89**	**99.17**	**82.51**	**91.21**	**96.55**	**98.29**	**98.86**	**99.14**
$ASPS_r$		81.79	90.74	96.36	98.18	98.79	99.09	82.46	91.14	96.52	98.27	98.85	**99.14**
$ASPS_m$		82.92	91.40	96.62	98.31	98.87	99.16	82.40	91.14	96.52	98.28	98.85	**99.14**
Original	$\tau = 5$	32.19	52.79	70.47	84.59	89.71	92.30	88.85	94.55	97.81	98.91	99.27	99.45
Reverse		83.30	88.71	93.13	95.87	97.24	97.94	89.38	94.81	97.91	98.96	99.31	99.48
Random		88.02	94.10	97.63	98.82	99.21	99.41	87.77	93.99	97.58	98.80	99.20	99.40
ICBP		88.37	94.28	97.70	98.86	99.24	99.43	88.77	94.50	97.79	98.90	99.27	99.45
inCTPri		90.47	95.34	98.13	99.07	99.38	99.53	90.69	95.44	98.17	99.09	99.39	99.54
$ASPS_e$		**91.11**	**95.66**	**98.25**	**99.13**	**99.42**	**99.57**	**91.16**	**95.68**	**98.26**	**99.14**	**99.42**	**99.57**
$ASPS_r$		90.04	95.13	98.04	99.03	99.35	99.51	90.69	95.45	98.17	99.09	99.39	99.54
$ASPS_m$		90.97	95.59	98.23	99.12	99.41	99.56	90.73	95.47	98.18	99.09	99.39	99.55

2) Does the ASPS method lead to higher fault detection rates than intuitive prioritization methods such as reverse prioritization and random prioritization? In this part, we compare the ASPS method with Reverse and Random, in terms of fault detection rate.

As shown in the tables, $ASPS_e$, $ASPS_r$, and $ASPS_m$ have higher NAPFD values than Reverse in 76.32% (145/190), 76.84% (146/190), and 78.42% (149/190) of cases, respectively. Furthermore, these three ASPS methods outperform Random in 97.37% (185/190), 95.79% (182/190), and 94.74% (180/190) of cases, respectively. Also, as the values of p increase, the improvement of the ASPS method over Reverse or Random decreases.

However, there are cases where either Reverse or Random obtain interaction test sequences with the highest NAPFD values. For instance, for subject program count with $\tau = 4$, Reverse performs best among all prioritization strategies, regardless of p value or interaction test suite construction tool (ACTS or PICT); likewise, for subject program flex with ACTS at $\tau = 2$, when $p = 75\%$ or 100%, Random obtains interaction test sequences with the highest NAPFD values.

In conclusion, in 76.32% ~ 97.37% of cases, the ASPS method performs better than the two intuitive prioritization methods Reverse and Random, according to NAPFD. Furthermore, similar to Original, the ASPS method favors the cases where smaller percentages of interaction test sequence are executed, compared with Reverse and Random.

3) Is the ASPS method better than "fixed-strength prioritization" in terms of fault detection rates? In this part, we compare fault detection rates of interaction test sequences prioritized by the ASPS method against two implementations of "*fixed-strength prioritization*", ICBP and inCTPri.

For subject program flex at strength $\tau = 2$, ICBP performs better than the ASPS method for some p values; otherwise, the ASPS method has higher NAPFD values, regardless of p value. More specifically, $ASPS_e$, $ASPS_r$, and $ASPS_m$ have higher rates of fault detection than ICBP in 85.79% (163/190), 71.58% (136/190), and 86.31% (164/190) of cases,

respectively. The main reason for this phenomenon is that ICBP is an implementation of "*fixed-strength prioritization*", which means that each selected element is evaluated according to a fixed strength τ. In addition, ICBP does not actually change the prioritization strength value throughout the prioritization process, and may consequently detect faults with the FTFI number = τ more quickly than ASPS, and faults with the FTFI number < τ more slowly, because the ASPS method focuses on aggregate strengths.

According to the NAPFD values shown in tables, ASPS_e, ASPS_r, and ASPS_m have better performances than inCTPri, in 70.00% (133/190), 43.16% (82/190), and 70.53% (134/190) of cases, respectively. The improvements of the ASPS methods over inCTPri in medium-sized programs (flex and grep) are larger than those in small-sized programs (count and nametbl). Additionally, as the values of p increase, the improvement of the ASPS method against inCTPri generally decreases. Similar to ICBP, inCTPri is an implementation of "*fixed-strength prioritization*", which means that it also prioritizes each test case using a fixed strength. However, they have different performances, something which can be explained as follows: ICBP uses a fixed strength τ throughout the prioritization process, but inCTPri uses different strengths (2, 3, $\cdots$, and τ) to guide the interaction test suite prioritization.

Both ICBP and inCTPri use only one strength to prioritize each combinatorial test case, while ASPS uses different strengths when choosing each, which could explain why it performs better in most cases. On the other hand, although neither ICBP nor inCTPri consider different strengths to guide the selection of each test case, the selected best candidate *tc* also involves additional information on interaction coverage at higher strengths. For example, suppose that *tc* covers a number of uncovered λ-wise value combinations, then according to the proof of Theorem 1, it can be concluded that *tc* also covers a number of uncovered value combinations at strengths higher than λ. Furthermore, since the fault distribution of each subject program (the FTFI number) is unknown before testing, it is

Table 10: The NAPFD metric (%) for different prioritization techniques for subject program flex when executing the percentage of interaction test sequence.

Method	Strength	p of ACTS Interaction Test Sequence Executed						p of PICT Interaction Test Sequence Executed					
		5%	10%	25%	50%	75%	100%	5%	10%	25%	50%	75%	100%
Original	$\tau = 2$	5.88	12.50	34.56	44.11	55.88	64.27	8.82	16.91	53.92	72.40	77.63	81.28
Reverse		13.24	27.94	48.53	64.82	72.94	76.91	7.35	16.91	48.53	66.74	72.45	76.75
Random		15.68	27.39	51.69	67.61	74.18	77.69	15.03	27.48	53.25	69.58	75.82	79.87
ICBP		15.12	31.10	62.96	**76.10**	**80.29**	**82.35**	14.94	29.69	**61.31**	75.12	79.62	82.66
inCTPri		15.12	31.10	62.96	**76.10**	**80.29**	**82.35**	14.94	29.69	**61.31**	75.12	79.62	82.66
ASPS_e		**16.41**	**32.28**	**63.19**	75.96	80.19	82.27	16.59	28.25	60.11	74.82	79.60	82.67
ASPS_r		15.41	30.31	62.24	75.37	79.75	81.94	**17.35**	**30.57**	59.81	75.05	**79.93**	**82.94**
ASPS_m		15.44	31.32	62.60	75.73	80.05	82.17	14.41	29.26	61.29	**75.31**	79.79	82.77
Original	$\tau = 3$	26.96	45.83	66.82	77.27	81.00	83.96	22.55	46.81	72.61	80.42	83.03	85.00
Reverse		29.41	42.89	56.34	71.88	79.11	82.98	**42.65**	58.09	75.83	**83.50**	**86.06**	**87.40**
Random		36.35	52.77	72.59	81.51	84.92	87.10	36.17	51.43	71.02	79.99	83.36	85.29
ICBP		43.57	62.09	77.66	83.66	85.97	87.88	41.41	60.42	76.65	82.44	84.65	86.18
inCTPri		41.88	61.30	77.33	83.40	85.80	87.82	40.35	59.89	76.63	82.61	84.83	86.32
ASPS_e		**44.54**	**62.55**	77.61	83.62	85.92	87.84	38.26	60.39	**77.43**	82.84	85.04	86.52
ASPS_r		42.21	60.68	77.35	83.59	86.01	87.85	40.40	59.23	76.43	82.68	84.91	86.39
ASPS_m		43.50	62.33	**77.96**	**83.91**	**86.18**	**88.10**	42.09	**61.44**	77.21	82.77	84.81	86.23
Original	$\tau = 4$	32.11	51.70	68.20	78.28	83.79	87.90	41.67	49.88	72.43	80.33	83.79	86.43
Reverse		50.25	59.62	64.38	76.79	84.17	88.19	58.09	70.10	79.60	85.98	88.69	90.08
Random		51.75	68.36	81.31	87.77	**90.74**	**92.85**	52.52	67.10	80.16	85.49	87.88	89.36
ICBP		59.98	74.36	83.96	88.33	90.54	92.72	61.04	73.40	82.97	86.72	**88.86**	**90.10**
inCTPri		61.67	75.46	84.01	88.32	90.66	92.82	61.82	74.21	**83.37**	**86.93**	88.67	89.83
ASPS_e		62.11	75.83	**84.50**	**88.44**	90.50	92.69	60.63	73.66	83.00	86.41	88.44	89.68
ASPS_r		59.90	73.92	83.48	88.36	90.72	92.80	59.25	72.25	82.41	86.50	88.51	89.78
ASPS_m		**62.85**	**76.07**	84.29	88.30	90.31	92.52	**62.19**	**74.29**	83.26	86.60	88.67	89.94
Original	$\tau = 5$	35.96	57.54	74.23	82.64	87.48	90.63	55.15	71.15	82.30	89.06	91.80	93.86
Reverse		52.81	60.55	64.88	71.81	81.03	85.81	59.71	71.57	81.13	87.56	89.76	92.13
Random		65.23	77.11	86.13	90.77	**93.09**	94.72	61.14	74.23	84.26	89.01	91.52	93.36
ICBP		71.39	80.69	87.50	91.03	93.02	94.67	69.31	79.40	86.58	90.49	92.95	94.71
inCTPri		73.23	81.27	86.97	90.45	92.81	94.54	71.92	80.55	86.20	89.69	92.33	94.23
ASPS_e		73.36	81.71	**87.54**	91.02	**93.09**	**94.76**	70.84	80.52	**86.98**	**90.59**	**93.05**	**94.77**
ASPS_r		70.62	79.95	87.30	**91.06**	92.97	94.61	69.31	79.15	86.54	90.32	92.51	94.30
ASPS_m		**73.91**	**81.73**	87.02	90.48	92.74	94.44	**72.34**	**80.69**	86.53	90.27	92.66	94.49

Table 11: The NAPFD metric (%) for different prioritization techniques for subject program grep when executing the percentage of interaction test sequence.

Method	Strength	p of ACTS Interaction Test Sequence Executed						p of PICT Interaction Test Sequence Executed					
		5%	10%	25%	50%	75%	100%	5%	10%	25%	50%	75%	100%
Original	$\tau = 2$	25.00	32.50	37.27	41.96	51.18	60.54	25.00	32.50	41.82	66.74	74.71	78.62
Reverse		30.00	35.00	45.45	59.35	69.26	74.67	30.00	38.75	54.09	68.48	75.86	79.47
Random		33.03	44.30	60.15	70.35	75.34	78.75	32.78	45.09	62.85	74.23	79.33	82.04
ICBP		31.13	42.74	61.80	72.36	76.10	78.69	30.65	42.30	61.42	74.29	79.58	82.24
inCTPri		31.13	42.74	61.80	72.36	76.10	78.69	30.65	42.30	61.42	74.29	79.58	82.24
$ASPS_e$		**36.55**	**49.55**	**66.35**	**75.35**	**78.66**	**80.90**	31.05	44.35	64.70	76.23	80.89	83.22
$ASPS_r$		31.70	44.30	63.07	72.89	76.67	79.43	**34.05**	**46.48**	64.53	75.79	80.55	82.97
$ASPS_m$		33.15	46.25	64.97	74.03	77.57	80.05	33.45	45.73	**64.94**	**76.39**	**80.99**	**83.29**
Original	$\tau = 3$	37.86	44.33	61.58	76.84	84.56	88.50	50.00	62.00	77.43	86.00	90.63	93.00
Reverse		47.86	64.00	74.60	82.30	85.44	89.15	58.57	70.00	79.86	87.87	91.88	93.93
Random		57.17	67.65	78.22	84.87	88.47	90.99	58.11	69.50	80.58	87.28	90.93	93.18
ICBP		56.44	69.51	79.94	86.27	89.92	92.48	58.69	71.53	80.69	87.56	91.34	93.53
inCTPri		55.94	69.84	80.68	**87.42**	**91.10**	**93.37**	58.67	71.57	81.51	88.20	91.81	93.88
$ASPS_e$		59.30	71.21	80.81	86.65	90.45	92.88	58.33	72.07	**82.67**	88.88	**92.36**	**94.29**
$ASPS_r$		**60.17**	**72.20**	81.09	86.01	89.06	91.80	**60.17**	**72.39**	82.14	**89.05**	92.33	94.25
$ASPS_m$		59.57	72.23	**81.40**	86.51	89.98	92.52	58.12	71.65	81.88	88.62	92.13	94.12
Original	$\tau = 4$	43.93	55.86	70.54	82.89	88.57	91.44	61.07	68.62	78.72	84.36	86.24	89.53
Reverse		63.93	67.24	75.88	82.99	86.05	89.56	51.79	64.31	79.93	89.80	93.20	94.90
Random		65.96	75.25	84.43	89.89	92.65	94.45	66.06	75.33	85.23	90.82	93.42	95.01
ICBP		67.97	77.58	86.44	92.38	94.91	96.19	70.94	78.59	86.22	92.36	94.90	96.18
inCTPri		71.02	79.43	87.07	92.85	95.22	96.42	71.22	79.44	**87.74**	**93.13**	**95.42**	**96.57**
$ASPS_e$		70.64	78.17	86.24	92.26	94.83	96.13	**71.59**	78.98	86.81	92.59	95.06	96.30
$ASPS_r$		**71.44**	**79.62**	**87.94**	**93.39**	**95.58**	**96.69**	70.09	**79.74**	87.62	92.91	95.28	96.46
$ASPS_m$		70.24	78.49	86.96	92.75	95.15	96.37	70.98	79.13	86.48	92.40	94.93	96.20
Original	$\tau = 5$	45.95	48.02	49.22	59.43	71.96	78.97	66.19	78.26	87.98	93.99	95.99	97.00
Reverse		64.29	72.33	77.11	86.10	90.73	93.05	79.76	85.00	94.04	97.02	98.01	98.51
Random		70.05	77.18	85.43	90.72	93.50	95.11	70.05	77.18	85.43	90.72	93.50	95.11
ICBP		72.32	80.33	88.88	94.36	96.24	97.18	72.32	80.33	88.88	94.36	96.24	97.18
inCTPri		74.00	81.41	88.95	94.44	96.29	97.22	74.52	82.47	90.57	95.28	96.85	97.64
$ASPS_e$		**76.98**	82.92	90.33	95.10	96.73	97.55	75.89	81.84	89.18	94.43	96.29	97.22
$ASPS_r$		76.60	82.61	89.07	94.16	96.10	97.08	75.30	81.78	89.52	94.50	96.34	97.25
$ASPS_m$		76.49	**83.69**	**91.09**	**95.47**	**96.98**	**97.73**	**76.49**	**83.69**	**91.09**	**95.47**	**96.98**	**97.73**

reasonable that ICBP and inCTPri occasionally have better fault detection rates than ASPS (especially $ASPS_r$, as its weighting distribution is assigned in a random manner).

In conclusion, according to the fault detection rates (the NAPFD values), ASPS performs better than both "*fixed-strength prioritization*" implementations, that is, ICBP (in 71.58% ~ 86.32% of cases) and inCTPri (in about 70.00% of cases for $ASPS_e$ and $ASPS_m$; and in 43.16% of cases for $ASPS_r$).

4) Which weighting distribution for the ASPS method gives the best fault detection rates? In this part, we study the fault detection rate of the ASPS method with three weighting distributions, so as to determine which weighting distribution is best.

From the experimental data, no weighting distribution is always best, because each weighting distribution performs best for some cases, but worst for others. Consider subject program grep (Table 8), for example: (a) ASPS with equal weighting distribution ($ASPS_e$) has the best NAPFD values for ACTS at strength $\tau = 2$; (b) ASPS with random weighting distribution ($ASPS_r$) performs best for ACTS at strength $\tau = 4$; and (c) ASPS with empirical FTFI percentage weighting distribution ($ASPS_m$) has the best fault detection at strength $\tau = 5$, for both ACTS and PICT.

On the whole, however, equal and empirical FTFI percentage weighting distributions have better rates of fault detection than random weighting distributions. As shown in Fig. 2, $ASPS_e$ achieves the best NAPFD values in 48.33% (91.83/190) of cases, followed by $ASPS_m$ in 34.65% (65.83/190) of cases. In other words, for the ASPS method, equal and empirical FTFI percentage weighting distributions would be the better choice in practical testing.

In conclusion, although each weighting distribution can perform best in some cases, the equal and empirical FTFI percentage weighting distributions perform better than random weighting distribution.

5) Time-cost analysis: In this part, we further investigate

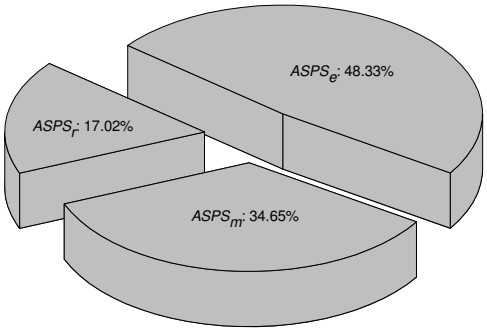

Figure 2: Comparison of $\texttt{ASPS}_e$, $\texttt{ASPS}_r$, and $\texttt{ASPS}_m$ according to NAPFD.

the time cost of the ICBP, inCTPri, and ASPS[11] methods, to help guide practical use.

Table 12 presents the time cost (in seconds) for different prioritization techniques of interaction test suites on different subject programs. From the experimental data, we can observe that our method ASPS has very similar time cost to ICBP. It can also be observed that when the strength $\tau = 2$, the time cost of ASPS is similar to that of inCTPri; but as the strength τ increases ($\tau = 3, 4, 5$), the time cost of ASPS differs from that of inCTPri. More specifically, for the two small-sized programs (count and nametbl), ASPS requires less time, but for the medium-sized programs (flex and grep), inCTPri has lower time costs.

According to the fault detection rates, ASPS generally has better performance than ICBP and inCTPri, especially when the percentage of interaction test sequence executed (p) is lower, in which case, the prioritization time cost of ASPS is also lower. In other words, when testing resources are limited, ASPS should be chosen as the prioritization method; however, when testing resources are less constrained, inCTPri would be a better choice because it requires less prioritization time for medium-sized programs.

6) Conclusion: The experimental study using real programs shows that although each method may sometimes obtain the highest NAPFD values, the ASPS method (regardless of $\texttt{ASPS}_e$, $\texttt{ASPS}_r$, and $\texttt{ASPS}_m$) performs better than Original, Reverse, Random, and ICBP in at least 70% of cases. Furthermore, the ASPS method (except for $\texttt{ASPS}_r$) also has better fault detection rates than inCTPri in about 70% of cases. Additionally, equal and empirical FTFI percentage weighting distributions give better performance for ASPS than random weighting distribution. With regard to the time cost for different prioritization strategies, the ASPS method has similar performance to ICBP, and higher costs than inCTPri for some subject programs but lower for others.

5.3. Threats to validity

In spite of our best efforts, our experiments may face three threats to validity:

(1) The experimental setup: The experimental programs are well-coded and tested. We have tried to manually cross-validate our analyzed programs on small examples, and we are confident of the correctness of the experimental and simulation set-up.

(2) The selection of experimental data: Two commonly-used tools (ACTS and PICT) were used for generating different covering arrays at different strength values. Although they are used in the field of combinatorial interaction testing, both of them use greedy algorithms. Only four test profiles were used for the simulations, which, although representative, were limited. For the real-life experiments, we examined only four subject programs, two of which were of a relatively small size. To address these potential threats, additional studies will be conducted in the future using many other test profiles, a greater number of subject programs, and different algorithms for covering array construction such as simulated annealing based algorithms [8, 15].

(3) The evaluation of experimental results: In order to objectively evaluate the effectiveness of the proposed method,

[11] We do not consider Original, Reverse, and Random, because the cost of these methods should be much less than that of the methods which make use of some additional information (ICBP, inCTPri, and ASPS). Furthermore, since the three implementations of ASPS ($\texttt{ASPS}_e$, $\texttt{ASPS}_r$, and $\texttt{ASPS}_m$) have similar prioritization time, we use ASPS to represent all three.

Table 12: Time comparisons (in seconds) of different prioritization techniques of interaction test suites.

Subject Program	Prioritization Strategy	ACTS				PICT			
		$\tau = 2$	$\tau = 3$	$\tau = 4$	$\tau = 5$	$\tau = 2$	$\tau = 3$	$\tau = 4$	$\tau = 5$
count	ICBP	0.04	0.25	1.30	3.11	0.04	0.26	1.47	3.49
	inCTPri	0.04	0.22	1.34	5.48	0.04	0.24	1.59	6.15
	ASPS	0.04	0.25	1.30	3.12	0.04	0.26	1.47	3.51
nametbl	ICBP	0.07	0.49	2.27	3.65	0.07	0.44	2.35	3.65
	inCTPri	0.07	0.45	2.74	8.36	0.07	0.45	2.88	8.71
	ASPS	0.07	0.49	2.27	3.66	0.07	0.44	2.35	3.66
flex	ICBP	0.13	1.57	10.42	38.64	0.12	1.50	10.72	34.61
	inCTPri	0.13	1.00	5.86	27.10	0.12	1.11	6.55	22.26
	ASPS	0.13	1.57	10.53	38.94	0.12	1.50	10.81	34.97
grep	ICBP	0.31	8.13	57.63	190.59	0.37	7.89	54.85	195.70
	inCTPri	0.31	5.54	31.80	92.27	0.37	5.25	32.81	99.66
	ASPS	0.31	8.19	58.32	191.87	0.37	7.89	55.40	196.94

the covering value combinations and the fault detection were measured with the APCC and NAPFD metrics, which are commonly used in the study of test case prioritization. Additionally, we only presented the averages for each prioritization strategy rather than a full statistical analysis, which is something we will prepare in the future.

6. Conclusion and Future Work

Combinatorial interaction testing [29] is a well-accepted testing technique, but due to often limited testing resources, prioritization of interaction test suites in combinatorial interaction testing has become very important. A dissimilarity measure to evaluate combinatorial test cases is introduced in this paper, based on which, a new pure prioritization strategy for interaction test suites is proposed, "*aggregate-strength prioritization*". Compared with traditional interaction coverage based test prioritization ("*fixed-strength prioritization*") [1, 2, 4–7, 18, 30–33, 37, 39, 40], the proposed method uses more information to guide prioritization of test suites. From the perspective of covering value combinations and fault detection, experimental results demonstrate that in most cases our method outperforms the test-case-generation prioritization, the reverse test-case-generation prioritization, and the random prioritization. Additionally, in most cases, our method has better performance than two implementations of "*fixed-strength prioritization*" while maintaining a similar time cost.

Similar to "*fixed-strength prioritization*", our prioritization strategy is not limited to conventional software. For example, event-driven software is a widely used category of software that takes sequences of events as input, alters state, and outputs new event sequences [4, 5]. It would be interesting to apply our strategy to different software including event-driven software in the future. Additionally, since the challenges of which weighting to be used and of whether to use fixed-strength or aggregate-strength prioritization strategy may depend on characteristics of the system under test, it would be useful, but challenging, to investigate the application scope of each prioritization strategy (including different weighting distributions).

The interaction test suite construction tool in our study, PICT [10], uses a greedy algorithm to generate (select) combinatorial test cases: it selects an element as the next test cases such that it covers the largest number of value combinations at a given strength τ (the largest $UVCD_\tau$). Since PICT considers $UVCD_\tau$ as the benefit for each combinatorial test case, PICT actually considers the prioritization during its the process of combinatorial test case generation. According to Qu's classification [32], therefore, PICT belongs to the category of *re-generation prioritization*. Although the simulation results (Section 5.1) indicate that the differences between PICT and other prioritization methods are small in terms of the APCC metric, the experimental results against real-life programs (Section 5.2) show that other prioritization methods can obtain higher fault detection rates than PICT, according to the NAPFD metric. In the future, it will be important and interesting for us to solve the problem: do interaction test sequences obtained by *re-generation prioritization* need to be further ordered by the *pure prioritization* category?

Acknowledgements

The authors would like to thank Christopher M. Lott for providing us the source code and failure reports for count and nametbl, and the Software-artifact Infrastructure Repository (SIR) [11] for providing the source code and fault data for flex and grep. We would also like to thank D. Richard Kuhn for providing us the ACTS tool, and Justyna Petke for helping in the experimental setup. Additionally, we would like to acknowledge T. Y. Chen for the many helpful discussions and comments. This work is partly supported by the National Natural Science Foundation of China (Grant No. 61202110), the Natural Science Foundation of Jiangsu Province (Grant No. BK2012284), and the Senior Personnel Scientific Research Foundation of Jiangsu University (Grant No. 14JDG039).

References

[1] Bryce, R. C., Colbourn, C. J., 2005. Test prioritization for pairwise interaction coverage. In: Proceedings of the 1st International Workshop on Advances in Model-based Testing (A-MOST'05). pp. 1–7.

[2] Bryce, R. C., Colbourn, C. J., 2006. Prioritized interaction testing for pairwise coverage with seeding and contraints. Information and Software Technology 48 (10), 960–970.

[3] Bryce, R. C., Colbourn, C. J., Cohen, M. B., 2005. A framework of greedy methods for constructing interaction test suites. In: Proceedings of the 27th International Conference on Software Engineering (ICSE'05). pp. 146–155.

[4] Bryce, R. C., Memon, A. M., 2007. Test suite prioritization by interaction coverage. In: Proceedings of the Workshop on Domain Specific Approaches to Software Test Automation (DoSTA'07). pp. 1–7.

[5] Bryce, R. C., Sampath, S., Memon, A. M., 2011. Developing a single model and test prioritization strategies for event-driven software. IEEE Transactions on Software Engineering 37 (1), 48–64.

[6] Bryce, R. C., Sampath, S., Pedersen, J. B., Manchester, S., 2011. Test suite prioritization by cost-based combinatorial interaction coverage. International Journal of Systems Assurance Engineering and Management 2 (2), 126–134.

[7] Chen, X., Gu, Q., Zhang, X., Chen, D., 2009. Building prioritized pairwise interaction test suites with ant colony optimization. In: Proceedings of the 9th International Conference on Quality Software (QSIC'09). pp. 347–352.

[8] Cohen, M. B., Gibbons, P. B., Mugridge, W. B., Colbourn, C. J., 2003. Constructing test suites for interaction testing. In: Proceedings of the 25th International Conference on Software Engineering (ICSE '03). pp. 38–48.

[9] Cohen, M. B., Gibbons, P. B., Mugridge, W. B., Colbourn, C. J., Collofello, J. S., 2003. Variable strength interaction testing of components. In: Proceedings of the 27th Annual International Conference on Computer Software and Applications (COMPSAC'03). pp. 413–418.

[10] Czerwonka, J., 2006. Pairwise testing in real world: Practical extensions to test case generators. In: Proceedings of the 24th Pacific Northwest Software Quality Conference (PNSQC'06). pp. 419–430.

[11] Do, H., Elbaum, S. G., Rothermel, G., 2005. Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. Empirical Software Engineering 10 (4), 405–435.

[12] Elbaum, S., Malishevsky, A., Rothermel, G., 2001. Incorporating varying test costs and fault severities into test case prioritization. In: Proceedings of the 23rd International Conference on Software Engineering (ICSE'01). pp. 329–338.

[13] Elbaum, S., Malishevsky, A. G., Rothermel, G., 2002. Test case prioritization: A family of empirical studies. IEEE Transaction on Software Engineering 28 (2), 159–182.

[14] Fouché, S., Cohen, M. B., Porter, A., 2009. Incremental covering array failure characterization in large configuration spaces. In: Proceedings of the 18th International Symposium on Software Testing and Analysis (ISSTA'09). pp. 177–188.

[15] Garvin, B. J., Cohen, M. B., Dwyer, M. B., February 2011. Evaluating improvements to a meta-heuristic search for constrained interaction testing. Empirical Software Engineering 16 (1), 61–102.

[16] Ghandehari, L. S. G., Lei, Y., Xie, T., Kuhn, R., Kacker, R., 2012. Identifying failure-inducing combinations in a combinatorial test set. In: Proceedings of the 5th International Conference on Software Testing, Verification and Validation (ICST'12). pp. 370–379.

[17] Grindal, M., Lindström, B., Offutt, J., Andler, S. F., 2006. An evaluation of combination strategies for test case selection. Empirical Software Engineering 11 (4), 583–611.

[18] Huang, R., Chen, J., Li, Z., Wang, R., Lu, Y., 2014. Adaptive random prioritization for interaction test suites. In: Proceedings of the 29th Symposium On Applied Computing (SAC'14), To appear.

[19] Huang, R., Chen, J., Zhang, T., Wang, R., Lu, Y., 2013. Prioritizing variable-strength covering array. In: Proceedings of the IEEE 37th Annual Computer Software and Applications Conference (COMPSAC'13). pp. 502–511.

[20] Huang, R., Xie, X., Chen, T. Y., Lu, Y., 2012. Adaptive random test case generation for combinatorial testing. In: Proceedings of the IEEE 36th Annual Computer Software and Applications Conference (COMPSAC'12). pp. 52–61.

[21] Huang, Y.-C., Peng, K.-L., Huang, C.-Y., 2012. A history-based cost-cognizant test case prioritization technique in regression testing. Journal of Systems and Software 85 (3), 626–637.

[22] Jiang, B., Zhang, Z., Chan, W. K., Tse, T. H., 2009. Adaptive random test case prioritization. In: Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering (ASE'09). pp. 233–244.

[23] Kuhn, D. R., Reilly, M. J., 2002. An investigation of the applicability of design of experiments to software testing. In: Proceedings of the 27th Annual NASA Goddard/IEEE Software Engineering Workshop (SEW-27'02). pp. 91–95.

[24] Kuhn, D. R., Wallace, D. R., Gallo, A. M., 2004. Software fault interactions and implications for software testing. IEEE Transaction on Software Engineering 30 (6), 418–421.

[25] Lei, Y., Kacker, R., Kuhn, D. R., Okun, V., 2008. Ipog/ipod: Efficient test generation for multi-way software testing. Software Testing, Verification, and Reliability 18 (3), 125–148.

[26] Li, Z., Harman, M., Hierons, R., 2007. Search algorithms for regression test case prioritization. IEEE Transactions on Software Engineering 33 (4), 225–237.

[27] Lott, C., Rombach, H., 1996. Repeatable software engineering experiments for comparing defect-detection techniques. Empirical Software Engineering 1 (3), 241–277.

[28] Mei, L., Chan, W. K., Tse, T. H., Merkel, R. G., 2011. Xml-manipulating test case prioritization for xml-manipulating services. Journal of Systems and Software 84 (4), 603–619.

[29] Nie, C., Leung, H., 2011. A survey of combinatorial testing. ACM Computer Survey 43 (2), 11:1–11:29.

[30] Petke, J., Yoo, S., Cohen, M. B., Harman, M., 2013. Efficiency and early fault detection with lower and higher strength combinatorial interaction testing. In: Proceedings of the 12th Joint Meeting on European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE'13). pp. 26–36.

[31] Qu, X., Cohen, M. B., Rothermel, G., 2008. Configuration-aware regression testing: an empirical study of sampling and prioritization. In: Proceedings of the 17th International Symposium on Software Testing and Analysis (ISSTA'08). pp. 75–86.

[32] Qu, X., Cohen, M. B., Woolf, K. M., 2007. Combinatorial interaction regression testing: A study of test case generation and prioritization. In: Proceedings of the IEEE International Conference on Software Maintenance (ICSM'07). pp. 255–264.

[33] Qu, X., Cohen, M. B., Woolf, K. M., 2013. A study in prioritization for higher strength combinatorial testing. In: Proceedings of the 2nd International Workshop on Combinatorial Testing, (IWCT'13). pp. 285–294.

[34] Rothermel, G., Untch, R. H., Chu, C., Harrold, M. J., 2001. Prioritizing test cases for regression testing. IEEE Transactions on Software Engineering 27 (10), 929–948.

[35] Seroussi, G., Bshouty, N. H., 1988. Vector sets for exhaustive testing of logic circuits. IEEE Transactions on Information Theory 34 (3), 513–522.

[36] Srikanth, H., Banerjee, S., 2012. Improving test efficiency through system test prioritization. Journal of Systems and Software 85 (5), 1176–1187.

[37] Srikanth, H., Cohen, M. B., Qu, X., 2009. Reducing field failures in system configurable software: Cost-based prioritization. In: Proceedings of the 20th International Symposium on Software Reliability Engineering (ISSRE'09). pp. 61–70.

[38] Tai, K. C., Lei, Y., 2002. A test generation strategy for pairwise testing. IEEE Transaction on Software Engineering 28 (1), 109–111.

[39] Wang, Z., 2009. Test case generation and prioritization for combinatorial testing. Ph.D. thesis, Southeast University, Nanjing, Jiangsu, China.

[40] Wang, Z., Chen, L., Xu, B., Huang, Y., 2011. Cost-cognizant combinatorial test case prioritization. International Journal of Software Engineering and Knowledge Engineering 21 (6), 829–854.

[41] Wong, W. E., Horgan, J. R., London, S., Bellcore, H. A., 1997. A study of effective regression testing in practice. In: Proceedings of the 8th International Symposium on Software Reliability Engineering (ISSRE'97). pp. 264–274.

[42] Yilmaz, C., Cohen, M. B., Porter, A. A., 2006. Covering arrays for efficient fault characterization in complex configuration spaces. IEEE Transactions on Software Engineering 32 (1), 20–34.

[43] Yoon, H., Choi, B., 2011. A test case prioritization based on degree of risk exposure and its empirical study. International Journal of Software Engineering and Knowledge Engineering 21 (02), 191–209.

[44] Zhang, L., Hou, S.-S., Guo, C., Xie, T., Mei, H., 2009. Time-aware test-case prioritization using integer linear programming. In: Proceedings of the 18th International Symposium on Software Testing and Analysis (ISSTA'09). pp. 213–224.

[45] Zhang, Z., Zhang, J., 2011. Characterizing failure-causing parameter interactions by adaptive testing. In: Proceedings of the 20th International Symposium on Software Testing and Analysis (ISSTA'11). pp. 331–341.